SAND REPORT

SAND2008-6461
Unlimited Release
Printed November 2008

Xyce™ Parallel Electronic Simulator

Users' Guide, Version 4.1

Eric R. Keiter, Ting Mei, Thomas V. Russo, Eric L. Rankin, Roger P. Pawlowski, Richard L. Schiek, Keith R. Santarelli, Todd S. Coffey, Heidi K. Thornquist, Deborah A. Fixel

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/ordering.htm

SAND2008-6461
Unlimited Release
Printed November 2008

Xyce™ Parallel Electronic Simulator

Users' Guide, Version 4.1

Eric R. Keiter,
Ting Mei, Thomas V. Russo, Eric L. Rankin, Richard L. Schiek,
Keith R. Santarelli, Heidi K. Thornquist, and Deborah A. Fixel
Electrical and Microsystems Modeling

Todd S. Coffey
Roger P. Pawlowski
Applied Mathematics and Applications

Sandia National Laboratories
P.O. Box 5800
Mail Stop 0316
Albuquerque, NM 87185-0316

March 25, 2009

Abstract

This manual describes the use of the **Xyce** Parallel Electronic Simulator. **Xyce** has been designed as a SPICE-compatible, high-performance analog circuit simulator, and has been written to support the simulation needs of the Sandia National Laboratories electrical designers. This development has focused on improving capability over the current state-of-the-art in the following areas:

- Capability to solve extremely large circuit problems by supporting large-scale parallel computing platforms (up to thousands of processors). Note that this includes support for most popular parallel and serial computers.
- Improved performance for all numerical kernels (e.g., time integrator, nonlinear and linear solvers) through state-of-the-art algorithms and novel techniques.
- Device models which are specifically tailored to meet Sandia's needs, including some radiation-aware devices (for Sandia users only).
- Object-oriented code design and implementation using modern coding practices that ensure that the **Xyce** Parallel Electronic Simulator will be maintainable and extensible far into the future.

Xyce is a parallel code in the most general sense of the phrase - a message passing parallel implementation - which allows it to run efficiently on the widest possible number of computing platforms. These include serial, shared-memory and distributed-memory parallel as well as heterogeneous platforms. Careful attention has been paid to the specific nature of circuit-simulation problems to ensure that optimal parallel efficiency is achieved as the number of processors grows.

The development of **Xyce** provides a platform for computational research and development aimed specifically at the needs of the Laboratory. With **Xyce**, Sandia has an "in-house" capability with which both new electrical (e.g., device model development) and algorithmic (e.g., faster time-integration methods, parallel solver algorithms) research and development can be performed. As a result, **Xyce** is a unique electrical simulation capability, designed to meet the unique needs of the laboratory.

Acknowledgements

The authors would like to acknowledge the entire Sandia National Laboratories HPEMS (High Performance Electrical Modeling and Simulation) team, including Steve Wix, Carolyn Bogdan, Regina Schells, Ken Marx, Steve Brandon and Bill Ballard, for their support on this project.

Trademarks

Contacts

Bug Reports	http://joseki.sandia.gov/bugzilla http://charleston.sandia.gov/bugzilla
World Wide Web	http://xyce.sandia.gov http://charleston.sandia.gov/xyce
Email	xyce-support@sandia.gov

Contents

This page is left intentionally blank

Figures

This page is left intentionally blank

Tables

This page is left intentionally blank

1. Introduction

Welcome to **Xyce**

The **Xyce** Parallel Electronic Simulator is a SPICE-compatible [1] [2] circuit simulator, that has been written to support the unique simulation needs of electrical designers at Sandia National Laboratories. It is specifically targeted to run on large-scale parallel computing platforms but is also available on a variety of architectures including single processor workstations. It aims to support a variety of devices and models specific to Sandia needs as well as standard capabilities available from current commercial simulators.

1.1 Xyce Overview

The **Xyce** Parallel Electronic Simulator project was started in 1999 to support the simulation needs of electrical designers at Sandia National Laboratories. The current release of **Xyce** is version 4.1, and the code has evolved into a mature platform for large scale circuit simulation.

Xyce includes several unique features. In addition to allowing the simulation of circuits of unprecedented size, **Xyce** includes novel approaches to numerical kernels including time integration algorithms, nonlinear and linear solvers. The primary driver for this numerical innovation has been the need to simulate very large scale circuits (100,000 devices or more) on the analog level. However, it has yielded benefits, in terms of robustness and efficiency, for all classes of problems. Ideally, the increased numerical robustness minimizes the amount of simulation "tuning" required on the part of the designer.

1.2 Xyce Capabilities

Xyce has a number of unique features which are described in this section.

Support for large-scale parallel computing

Xyce is a truly parallel simulation code, designed and written from the ground up to support large-scale parallel computing architectures with up to thousands of processors. This gives **Xyce** the capability to solve circuit problems of unprecedented size in time frames that make these simulations practical.

Xyce as a parallel code uses a message passing parallel implementation, which allows it to run efficiently on the widest possible number of computing platforms. These include serial, shared-memory and distributed-memory parallel. Furthermore, careful attention has been paid to the specific nature of circuit-simulation problems to ensure that optimal parallel efficiency is achieved even as the number of processors grows (*parallel scaling*).

Improved performance for all numerical kernels

In writing **Xyce** from scratch, new algorithms and heuristics have been used which improve the overall performance of the various numerical kernels. For example, a number of new

developments have made it possible to reliably apply iterative linear solvers to circuit problems. This allows **Xyce** to scale well to much larger problem sizes than would be possible with a conventional circuit simulator. Using iterative linear solvers also allows **Xyce** to run much more effectively in parallel.

On the nonlinear solver level, the addition of continuation algorithms to **Xyce** has been another recent solver enhancement. In particular, **Xyce** has been very successful applying such algorithms to large MOSFET circuits. See chapter 8 for more details.

Device Model Support

New device models are continually being added to **Xyce** to meet the needs of Sandia users. For a complete description of each device, see the **Xyce** Reference Guide [3]. As there are many devices under development, several devices are available in the development branch of the code that are not available in the release branch. For current device availability, consult with the **Xyce** development team.

1.3 Reference Guide

A companion document, the **Xyce** Reference Guide [3], contains more detailed information about a number of topics. Included in this document is a netlist reference for the input-file commands and elements supported within **Xyce**; a command line reference, which describes the available command line arguments for **Xyce**; and quick-references for users of other circuit codes, such as Orcad's PSpice [4] and Sandia's ChileSPICE.

1.4 How to Use this Guide

This guide is designed so you can quickly find the information you need to use **Xyce**. It assumes that you are familiar with basic Unix-type commands, how Unix manages applications and files to perform routine tasks (e.g., starting applications, opening files and saving your work).

Typographical conventions

Before continuing in this Users' Guide, it is important to understand the terms and typographical conventions used. Procedures for performing an operation are generally num-

bered with the following typographical conventions.

Notation	Example	Description
`Verbatim text`	`> xmpirun -np 4`	Commands entered from the keyboard on the command line or text entered in a netlist. The initial > character is intended to represent the shell prompt.
Bold Roman Font	Set nominal temperature using the **TNOM** option.	SPICE-type parameters used in models, etc.
Gray Shaded Text	DEBUGLEVEL	Feature that is designed primarily for use by **Xyce** developers.
`[text in brackets]`	`Xyce [options] <netlist>`	Optional parameters.
`<text in angle brackets>`	`Xyce [options] <netlist>`	Parameters to be inserted by the user.
`<object with asterisk>*`	`K1 <ind. 1> [<ind. n>*]`	Parameter that may be multiply specified.
`<TEXT1|TEXT2>`	`.PRINT TRAN` `+ DELIMITER=<TAB|COMMA>`	Parameters that may only take specified values.

Table 1.1. Xyce typographical conventions.

1.5 Third Party License Information

Portions of the new DAE time integrator contained in the BackwardDifferentiation15 source and include files are derived from the IDA code from Lawrence Livermore National Laboratories and is licensed under the following license.

DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Additional BSD Notice
---

1. This notice is required to be provided under our contract with the U.S. Department of Energy (DOE). This work was produced at the University of California, Lawrence Livermore National Laboratory under Contract No. W-7405-ENG-48 with the DOE.

2. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately-owned rights.

3. Also, reference herein to any specific commercial products, process, or services by trade name, trademark, manufacturer or otherwise does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.

2. Installing and Running **Xyce**

Chapter Overview

This chapter describes the basic mechanics of installing and running **Xyce**. It includes the following sections:

- Section 2.1, **Xyce** *Installation*
- Section 2.2, *Running* **Xyce**

2.1 **Xyce** Installation

To obtain a copy of **Xyce**, contact the development team at `http://www.cs.sandia.gov/xyce`.

The installation procedure differs depending upon the operating system. Follow the instructions below to properly install **Xyce**.

Installing **Xyce** on UNIX

Xyce is installed from the command line. Users of Linux, BSD, IRIX, and other UNIX variants should follow this section. Examples are given for reference.

Instructions	Examples
Installation packages are named according the target operating system and architecture (parallel or serial).	`Install_Xyce_Linux.tar.gz` (serial) `Install_Xyce_Linux_OPENMPI.tar.gz` (parallel)
Unpack the appropriate package for your platform. A similarly named installation directory is then created.	`> tar xfz Install_Xyce_Linux.tar`
Enter this directory and run the installation shell script.	`> cd Install_Xyce_Linux` `> sh install_Linux.sh`
Provide the requested information.	`Where should Xyce be installed?` `/usr/local/Xyce-4.1`

We recommend that you specify a completely new directory in which to install Xyce rather than a general system directory. For example, `/usr/local/Xyce-4.1` would be a better choice than `/usr/local`. Doing this will help isolate your Xyce installation and make uninstallation or upgrade easier.

Installing **Xyce** on Microsoft Windows

The installation instructions here describe installation on a Windows system that does not have Cygwin installed. These instructions will need to be modified if Cygwin is installed, as the installation process includes a subset of Cygwin tools that will clash with the pre-installed versions. It is still possible to install **Xyce** *if Cygwin is installed, and the installer*

will give instructions to follow if it detects an existing Cygwin installation. Please contact the **Xyce** *development team for assistance if you encounter any problems installing in this case.*

The installation package is similarly named `Install_Xyce_Windows.zip`. This example illustrates the procedure for Windows XP Professional. Similar steps may be followed for other versions of Windows.

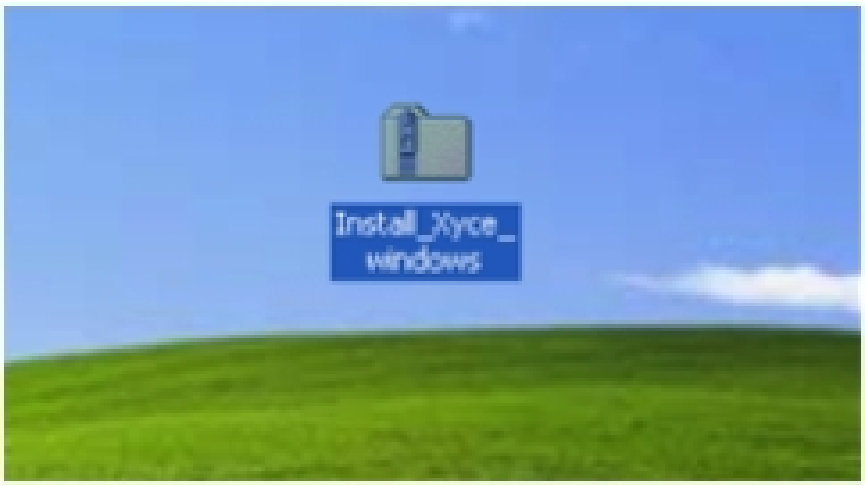

Figure 2.1. Obtain a copy of the installation package. Double click the file name to begin.

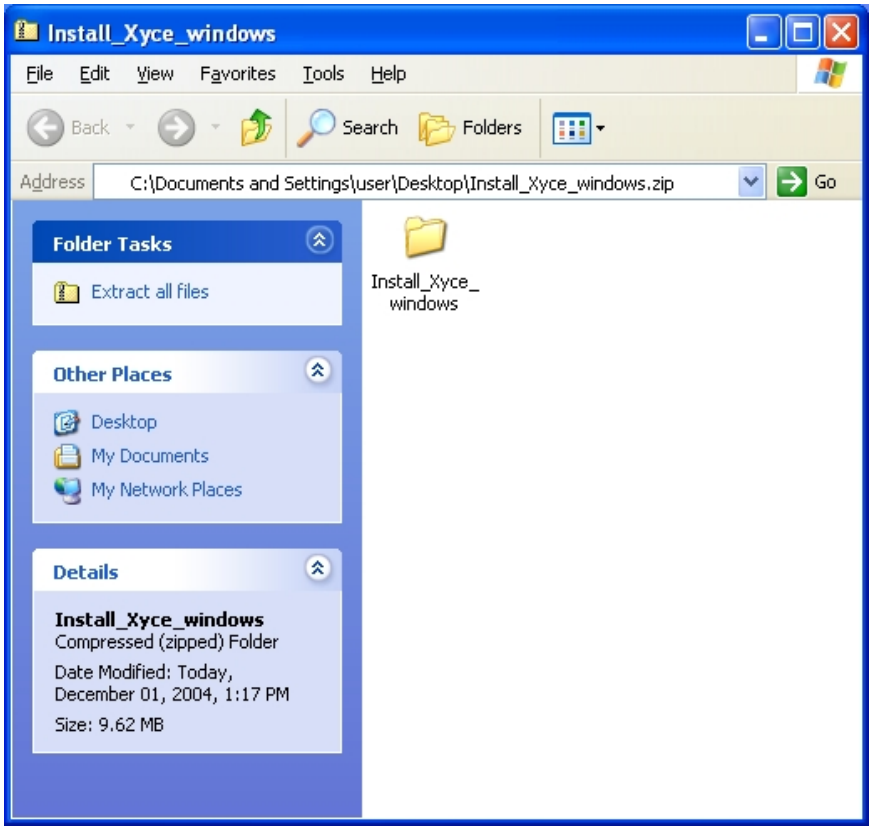

Figure 2.2. By default, Windows will display the contents of the zip file as shown above. Select **Extract all Files** to continue.

Users of programs such as *WinZip*, *PKZip*, *Winrar*, etc., may follow the decompression steps associated with those programs, and continue from Figure 2.6.

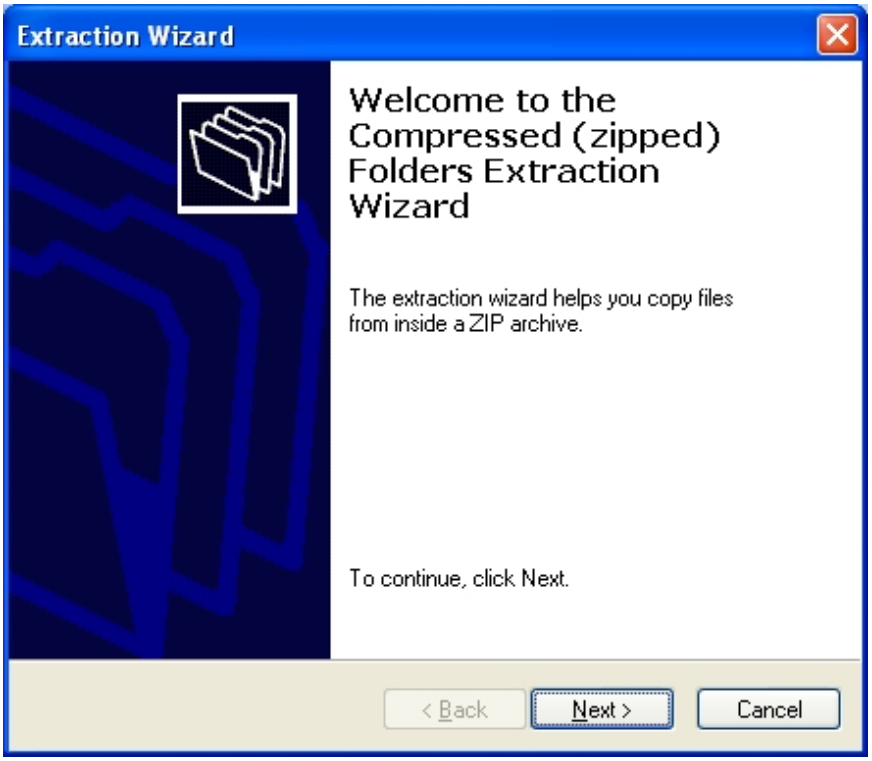

Figure 2.3. Click **Next** on the Extraction Wizard.

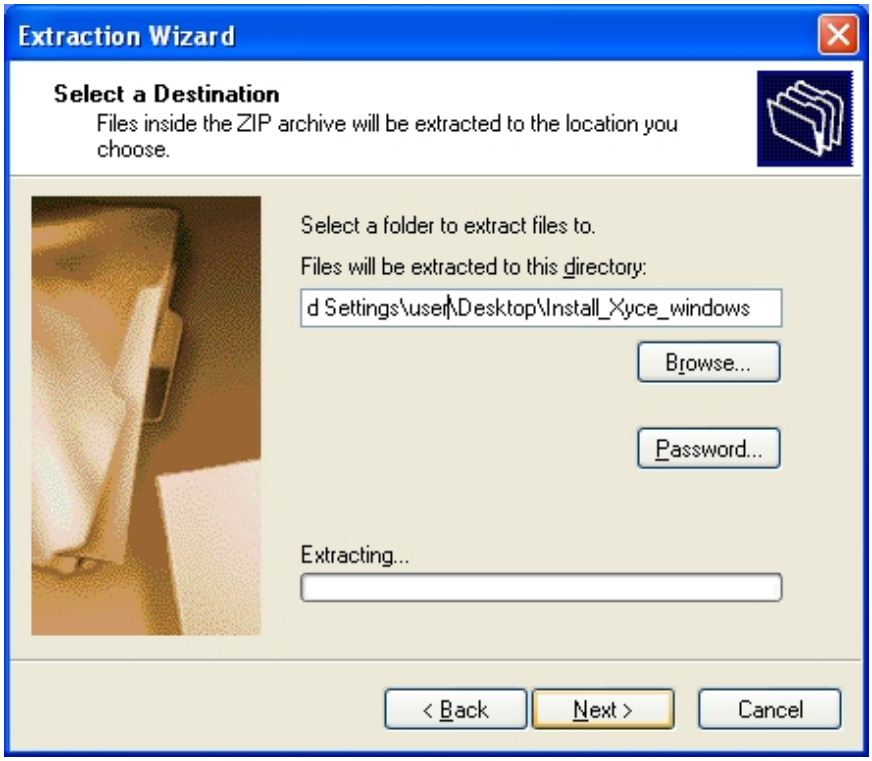

Figure 2.4. Select a destination for the archive contents and click **Next**.

We recommend that you specify a completely new directory in which to install Xyce rather than a general system directory. For example, `C:`**Xyce**`-4.1` would be a better choice than `C:`. Doing this will help isolate your Xyce installation and make uninstallation or upgrade easier.

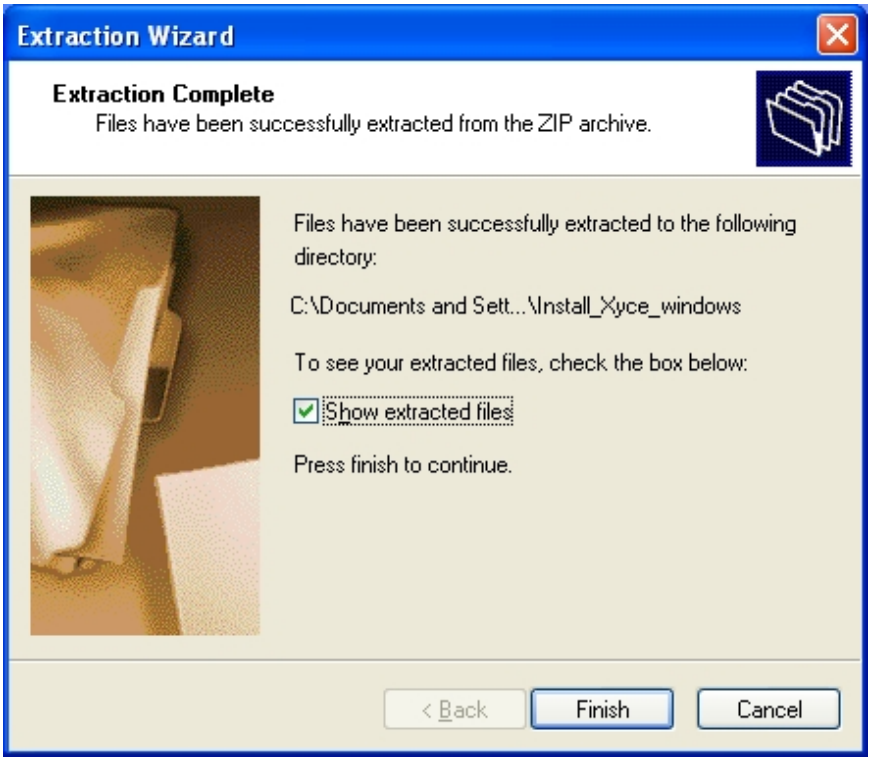

Figure 2.5. Select **Show extracted files** and click **Finish**.

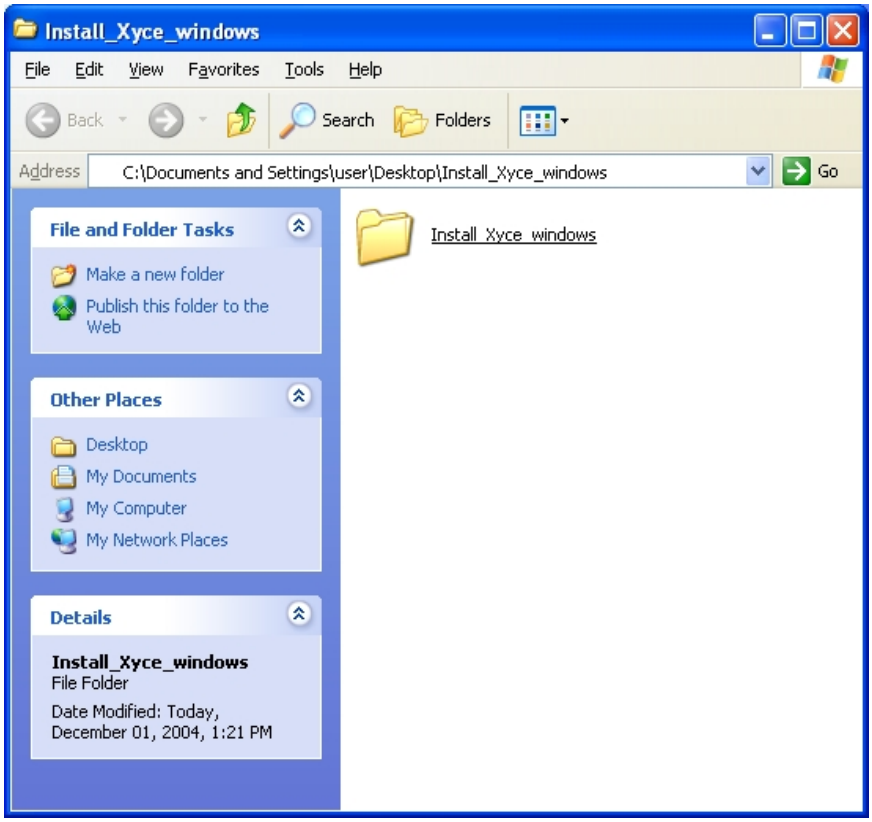

Figure 2.6. The contents of the extracted zip file are shown. Open the folder named **Install_Xyce_windows**.

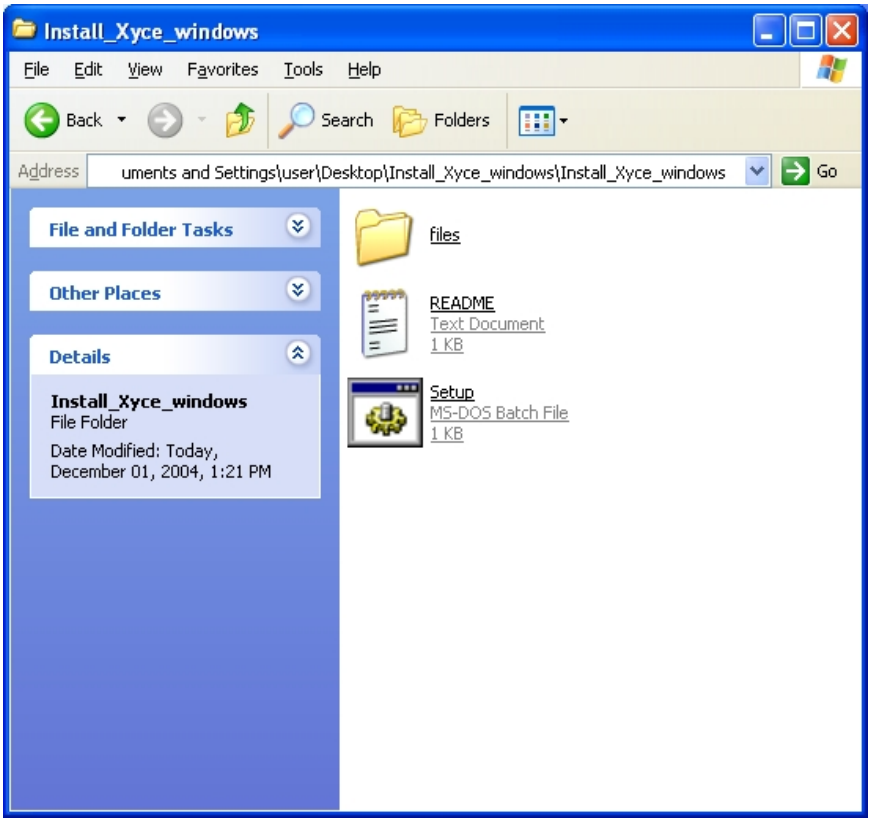

Figure 2.7. Double click **Setup** to begin.

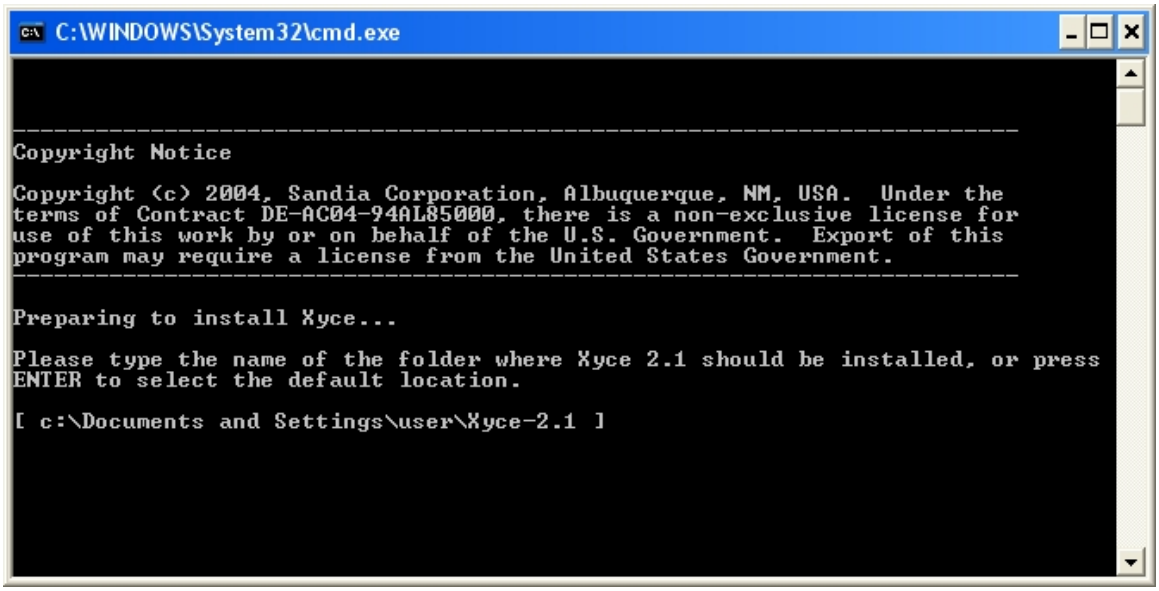

Figure 2.8. A command line window will open. Follow the on-screen instructions to proceed.

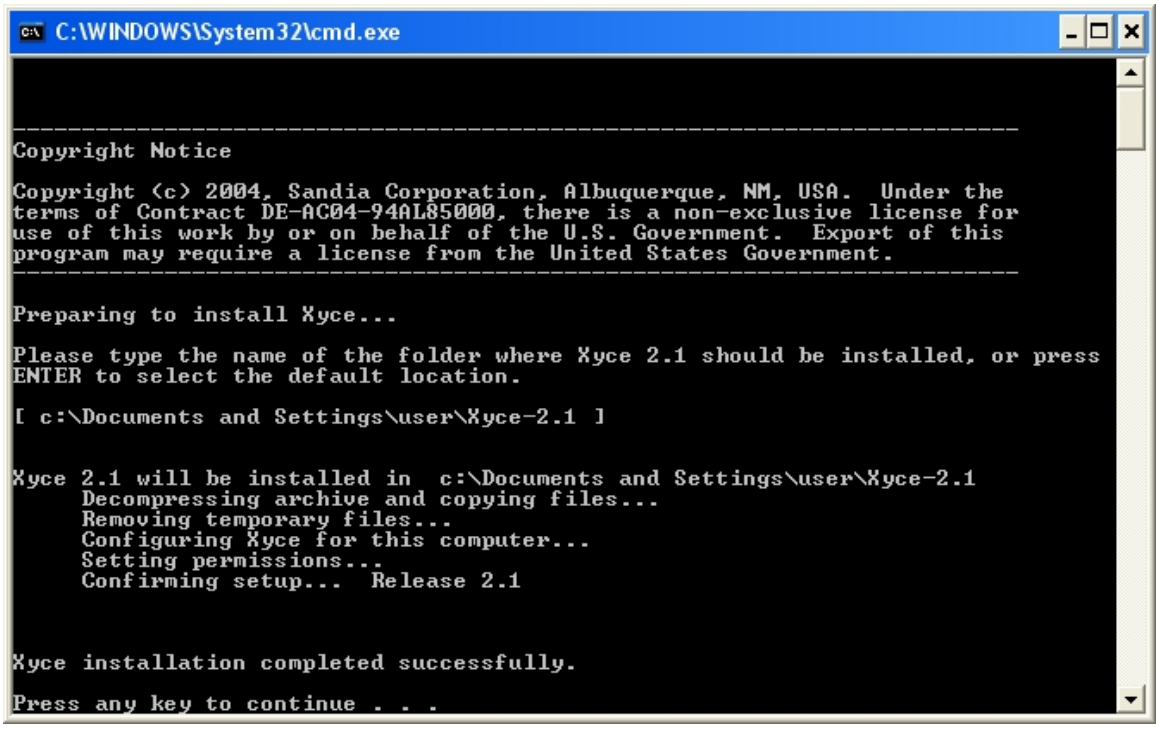

Figure 2.9. Status messages will display during setup. Confirmation of success will appear once all components are installed.

Important Notes

Completing the steps above will unpack **Xyce** to the specified directory. **IMPORTANT NOTE: if installing *both* serial and parallel versions of Xyce, you must specify different directories for each installation location. Failure to use different directories will cause the second installation to overwrite parts of the first and will likely yield an install that does not function.** Under the specified installation directories, the following subdirectories will be created:

- **bin** contains the executable used to start **Xyce**. The executable name will vary depending on the target operating system and architecture.
  - `runxyce` is the shell script for starting serial **Xyce** on Unix platforms.

  - `runxyce.bat` is the batch file for starting serial **Xyce** on Windows.
  - `xmpirun` is the wrapper script for mpirun used for running **Xyce** in parallel mode.

- **doc** contains the **Xyce** Users' Guide, comprehensive Reference Guide, and Release Notes. Read these for more information about this release and for detailed instructions on how to use **Xyce**.
- **lib** contains configuration files, libraries, and metadata for **Xyce**.
- **test** contains sample netlists and verification tools.

Uninstalling **Xyce**

Xyce comes with no special uninstall script.

The **Xyce** installation process simply unpacks a number of files into the directory you identify to the setup program. Removing those files uninstalls **Xyce**. If you followed our recommendation and installed to a completely separate directory like `/usr/local/Xyce-4.1`, uninstallation is as simple as removing the entire directory.

2.2 Running **Xyce**

While it is possible to connect **Xyce** to graphical interfaces, such as gEDA [5], this section only describes how **Xyce** is run from the command line, for both serial and MPI parallel simulations.

Command Line Simulation

Running **Xyce** from the command line is straightforward. The scripts `xmpirun` and `runxyce` set up the runtime environment and execute **Xyce**. *Help with accessing the command line on Microsoft Windows is available at the end of this chapter*. Depending on whether you are using a version compiled with MPI support or a serial version, there are two ways to begin running **Xyce**:

- Running serial **Xyce**:

```
> runxyce [options] <netlist filename>
```

- Running **Xyce** in parallel:

```
> xmpirun -np <# procs> [options] <netlist filename>
```

where [options] are the command line arguments for **Xyce**. For example, to log output to a file named sample.log type:

```
> runxyce -l sample.log <netlist filename>
```

The next example runs parallel **Xyce** on four processors and places the results into a comma separated value file named results.csv:

```
> xmpirun -np 4 -delim COMMA -o results.csv <netlist filename>
```

These examples assume that <netlist filename> is either in the current working directory or includes the path (full or relative) to the netlist file. Enclose the filename in quotation marks (" ") if the path contains spaces. Help is accessible with the -h option.

For MPI runs, [options] may also include command line arguments to mpirun. Consult the documentation installed with MPI on your platform for more details on MPI options. The -np <# procs> denotes the number of processors to use for the simulation. *NOTE: It is critical that the number of processors used is less than the number of devices and voltage nodes in the netlist.* The appropriate script used to run **Xyce** for each supported platform is listed in the Table 2.1.

Architecture	OS	Serial Executable	MPI Executable
x86	OSX	runxyce	xmpirun
x86 and x86-64	Linux		
x86	FreeBSD		
x86	Microsoft Windows	runxyce.bat	not available

Table 2.1. Platform scripts for running **Xyce**.

While **Xyce** is running, the progress of the simulation is output to the command line window.

Command Line Options

Xyce supports a handful of command line options which must be given *before* the netlist filename. The general usage is:

```
runxyce [options] <netlist filename>
```

Table 2.2 lists the available command line options.

Argument	Description	Usage	Default
-h	Help option. Prints usage and exits.	`-h`	-
-v	Prints the version banner and exits.	`-v`	-
-delim	Set the output file field delimiter.	`-delim <TAB|COMMA|string>`	-
-o	Place the results into specified file.	`-o <file>`	-
-l	Place the log output into specified file.	`-l <file>`	-
-r	Output a binary rawfile.	`-r <file>`	-
-a	Use with `-r` to output a readable (ascii) rawfile.	`-r <file> -a`	-
-nox	Use the NOX nonlinear solver.	`-nox <ON|OFF>`	on
-info	Output information on parameters.	`-info [device prefix] [level] [ON|OFF]`	-
-linsolv	Set the linear solver.	`-linsolv <KLU| SUPERLU|AZTECOO>`	klu(serial) and aztecoo(parallel)
-newdae	Use the new DAE time integrator.	`-newdae [ON|OFF]`	on
-param	Print a terse summary of model and/or device parameters.	`-param [<device prefix> [<level> [<INST|MOD>] ] ]`	-
-syntax	Check netlist syntax and exit.	`-syntax`	-
-norun	Netlist syntax and topology and exit.	`-norun`	-
-maxord	Maximum time integration order.	`-maxord <1..5>`	-
-method	Time integration method (old-dae only).	`-method <1..4>`	-
-gui	GUI file output.	`-gui`	-

Argument	Description	Usage	Default
-jacobian_test	Jacobian matrix diagnostic.	`-jacobian_test`	-

Table 2.2: List of **Xyce** command line arguments.

While these options are intended for general use, others may exist for new features that are disabled by default, and older features that are no longer supported. See the **Xyce** Reference Guide for a comprehensive list that also includes trial and deprecated options.

Running **Xyce** in Parallel

A parallel version of **Xyce** is available for several different platforms as shown in Table 2.1. Running **Xyce** in parallel requires the script `xmpirun` to be used with the appropriate parameters. For example, to run **Xyce** on two processors with an example netlist, type:

```
xmpirun -np 2 anExampleNetlist.cir
```

In general the number of processors is specified by using the `-np` argument to the appropriate `mpirun` command.

Guidance

This chapter has given the basic mechanics of running **Xyce** in parallel. For general guidance regarding solver options, partitioning options, and other parallel issues, refer to chapter 11. Distributed memory circuit simulation still contains a number of research issues, so obtaining an optimal simulation in parallel is a bit of an art.

Accessing the Microsoft Windows Command Line

Follow the steps below for help with accessing the command line on Windows XP Professional. Consult the operating system documentation for assistance with other versions of Windows.

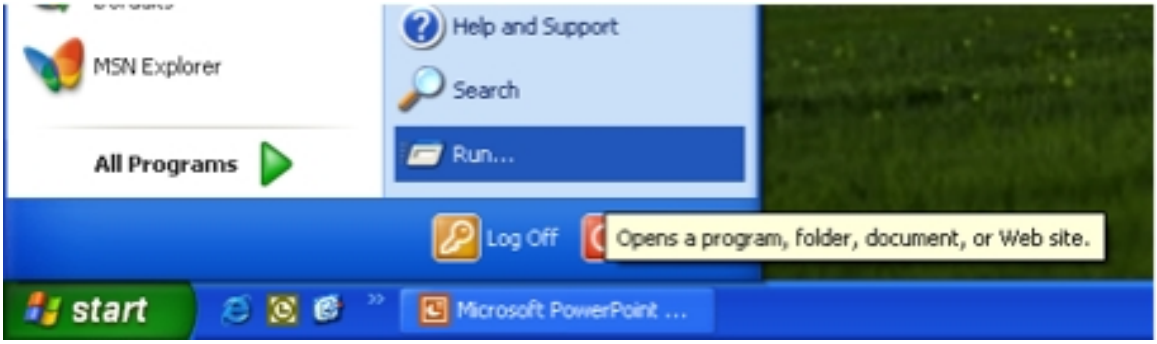

Figure 2.10. From the *Start* menu, click **Run...**.

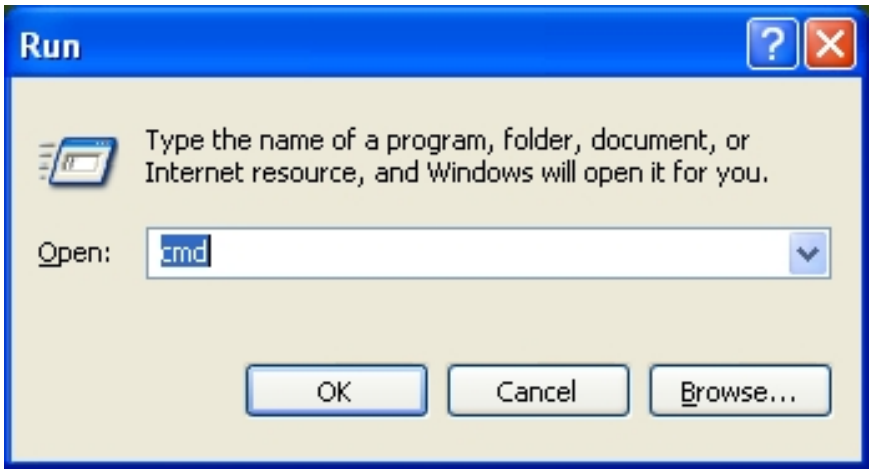

Figure 2.11. Type **cmd** and click **OK** to open the command line window.

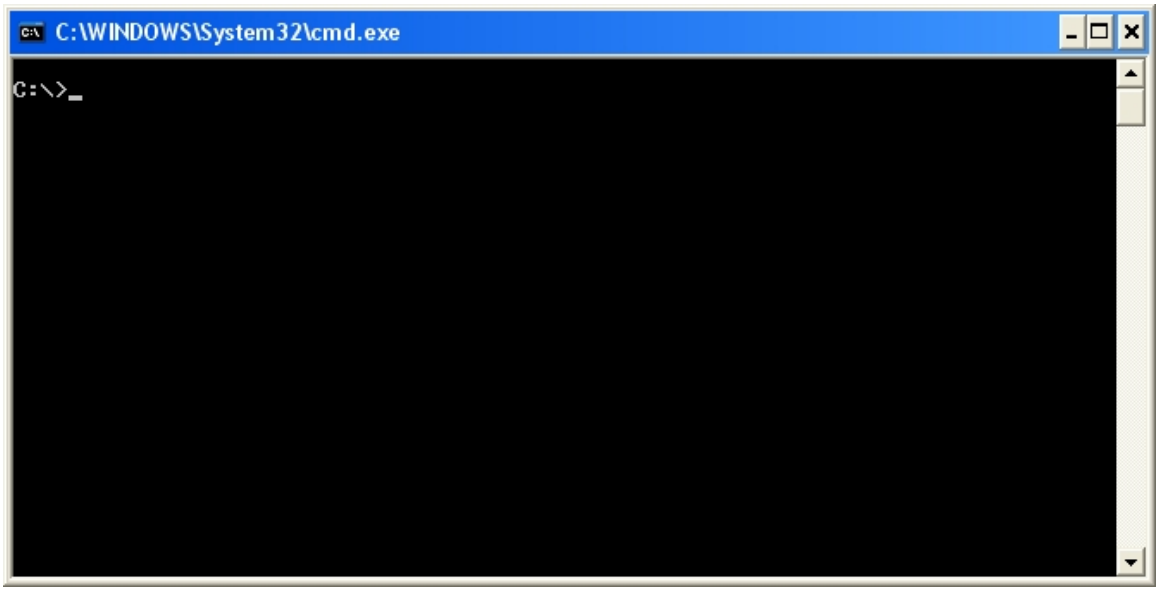

Figure 2.12. The command line window appears and is ready for use.

The following is an illustrated recap of the Command Line Simulation instructions provided earlier.

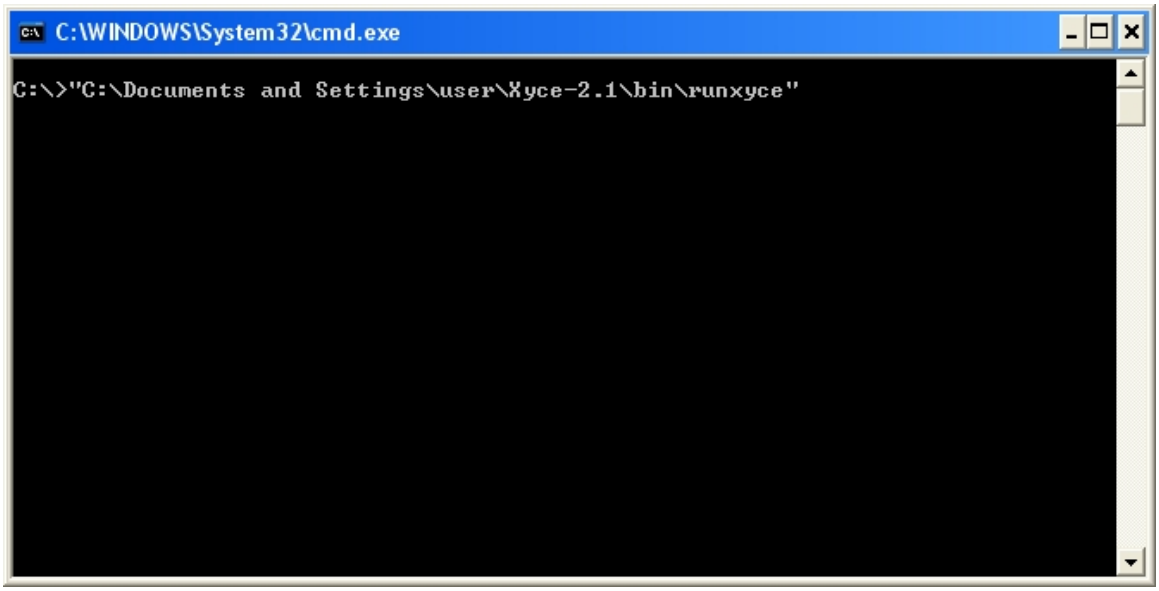

Figure 2.13. Type the full path to the **runxyce** file to execute **Xyce**. Note that the path must be enclosed in quotation marks if it contains spaces.

Figure 2.14. Using **runxyce** without any options or a netlist file-name displays a brief help menu.

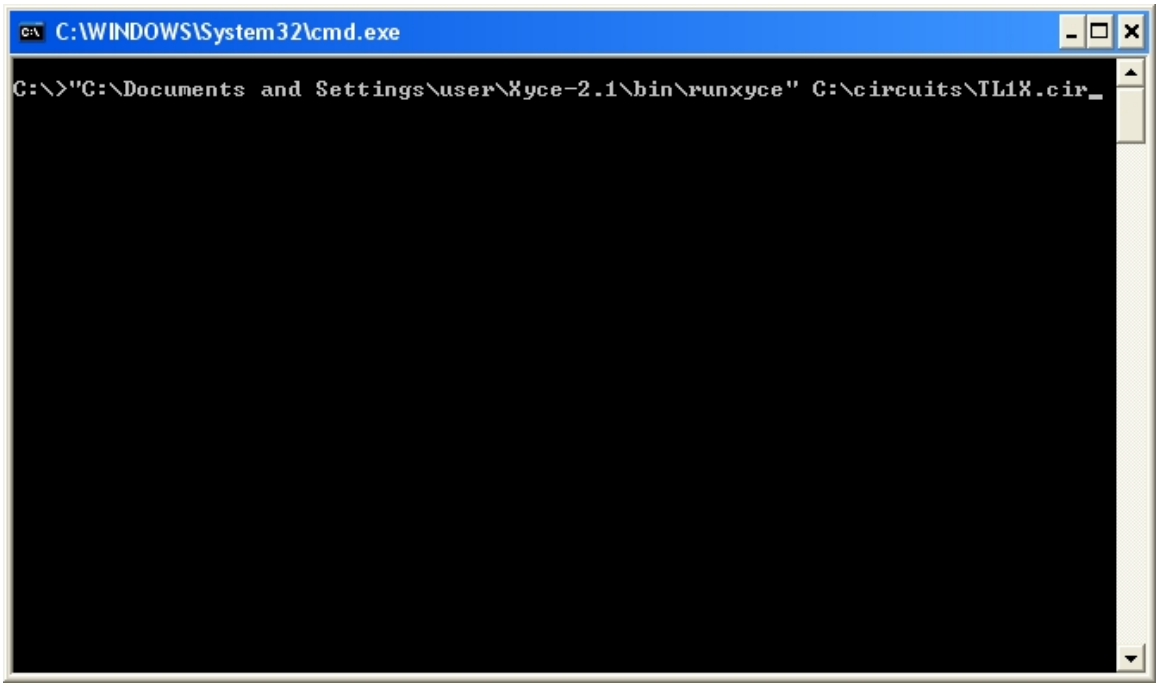

Figure 2.15. To begin a simulation type the path to **runxyce** followed by the netlist filename.

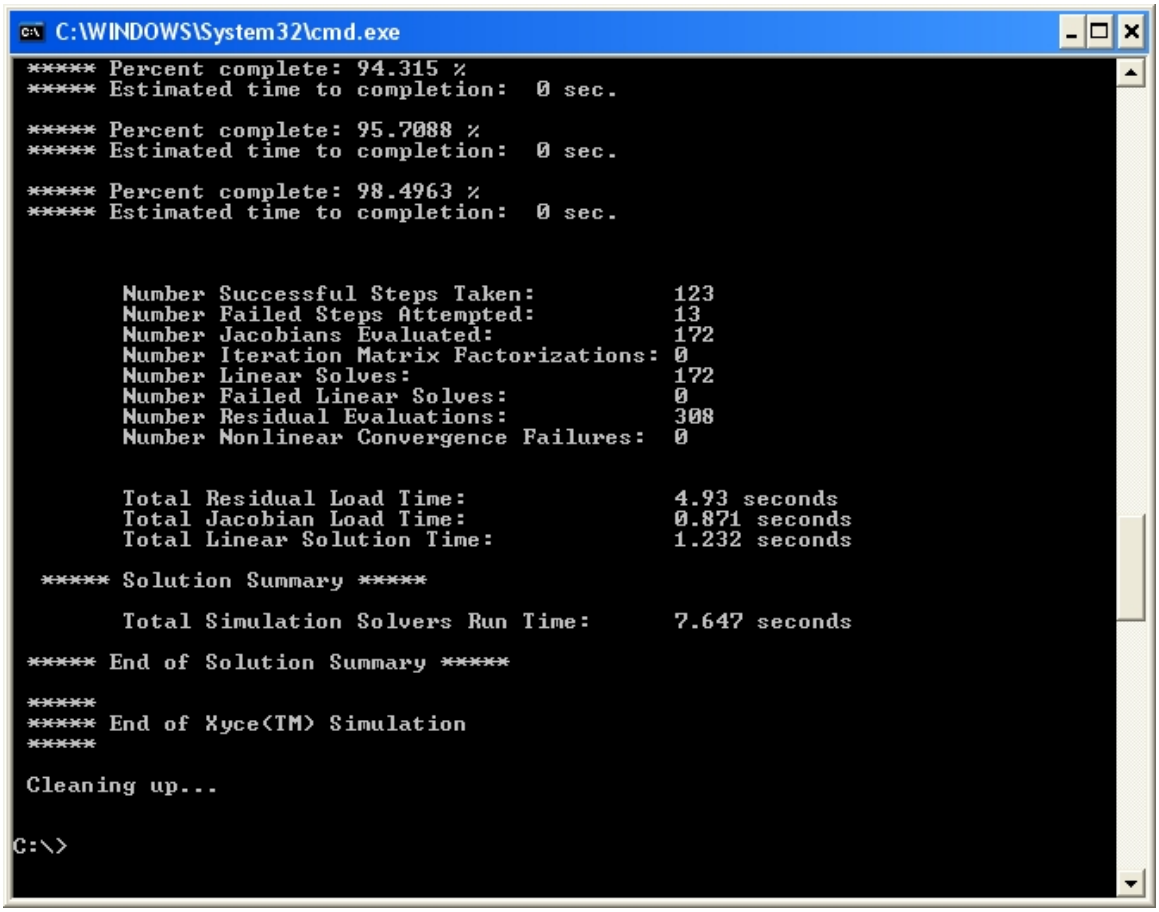

Figure 2.16. Output will scroll to the screen. Use **runxyce -h** for assistance with command line options.

This page is left intentionally blank

3. Simulation Examples with **Xyce**

Chapter Overview

This chapter provides several simple examples of **Xyce** usage. An example circuit is provided for each available analysis type.

3.1 Example Circuit Construction

This section describes how to use **Xyce** to create the simple diode clipper circuit shown in Figure 3.1.

While a schematic edit and capture capability is under development, **Xyce** currently only supports circuit creation via netlist editing. **Xyce** supports most of the standard netlist entries common to Berkeley SPICE 3F5 and Orcad PSpice. For users who are familiar with PSpice netlists, the differences between PSpice and **Xyce** netlists are listed in the **Xyce** Reference Guide [3].

Example: diode clipper circuit

1. Open a new netlist file using a standard text editor (*e.g.* VI, Emacs, Notepad, *etc.*).
2. Type the title on the first line of the netlist:

```
Diode Clipper Circuit
```

3. Create a 5V DC voltage source between nodes 1 and 0 by typing the following on a new line:

```
VCC 1 0 5V
```

4. Create another DC voltage source between nodes 3 and 0 by entering the following on a new line:

```
VIN 3 0 0V
```

5. Place the diodes in the circuit between nodes 2 and 1, and nodes 0 and 2, respectively, by entering the following lines:

```
D1 2 1 D1N3940
D2 0 2 D1N3940
```

6. Enter resistors R1, R2, R3 and R4, respectively:

```
R1 2 3 1K
R2 1 2 3.3K
```

```
R3 2 0 3.3K
R4 4 0 5.6K
```

7. Place the capacitor in the circuit:

```
C1 2 4 0.47u
```

8. Add the diode model to the netlist to complete it as Figure 3.2.

9. Complete the netlist by entering `.END` on the last line in the file. Save the file as `clipper.cir`. The complete netlist is shown in Figure 3.2 and the schematic in Figure 3.1.

The netlist in Figure 3.2 illustrates some of the syntax of a netlist input file. Netlists begin with a title (*e.g.* "`Diode Clipper Circuit`"), support comments (lines beginning with the "`*`" character), devices, model definitions and the "`.END`" statement.

This netlist file is not yet complete and will not run properly using **Xyce** (see Section 2.2 for instructions on running **Xyce**) as it lacks an analysis statement. As you proceed in this chapter, you will see how to add the appropriate analysis statement and run the clipper circuit.

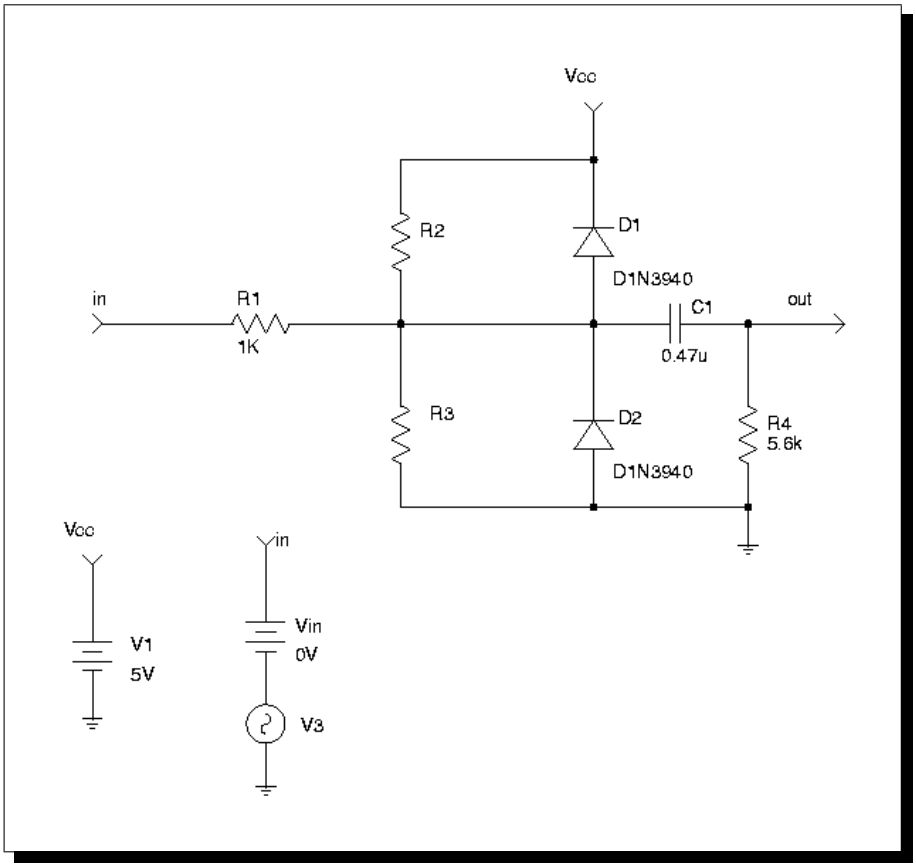

Figure 3.1. Schematic of diode clipper circuit with DC and transient voltage sources.

3.2 DC Sweep Analysis

In this section an example is given of DC sweep analysis using **Xyce**. The DC response of the clipper circuit is obtained by sweeping the DC voltage source (`Vin`) from -10 to 15 volts in 1 volt steps. For more details about DC analysis see Chapter 7.3 of this manual or the **Xyce** Reference Guide [3].

```
Diode Clipper Circuit
*
* Voltage Sources
VCC 1 0 5V
VIN 3 0 0V
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
*
* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE:  DIODE
* SUBTYPE:  RECTIFIER
.MODEL D1N3940 D(
+          IS = 4E-10
+          RS = .105
+           N = 1.48
+          TT = 8E-7
+         CJO = 1.95E-11
+          VJ = .4
+           M = .38
+          EG = 1.36
+         XTI = -8
+          KF = 0
+          AF = 1
+          FC = .9
+          BV = 600
+         IBV = 1E-4)
*
.END
```

Figure 3.2. Diode clipper circuit netlist.

Example: DC sweep analysis

To set up and run a DC sweep analysis using the diode clipper circuit:

1. Open the diode clipper circuit netlist file (`clipper.cir`) using a standard text editor (*e.g.* VI, Emacs, Notepad, *etc.*).
2. Enter the analysis control statement in the netlist:

```
.DC VIN -10 15 1
```

3. Enter the output control statement:

```
.PRINT DC V(3) V(2) V(4)
```

4. Save the netlist file and run **Xyce** on the circuit. For example, to run serial **Xyce**:

```
> runxyce clipper.cir
```

5. Open the results file (clipper.cir.prn) and examine (or plot) the output voltages that were calculated for nodes 3 (Vin), 2 and 4 (Out). Figure 3.4 shows the output plotted as a function of the swept variable Vin.

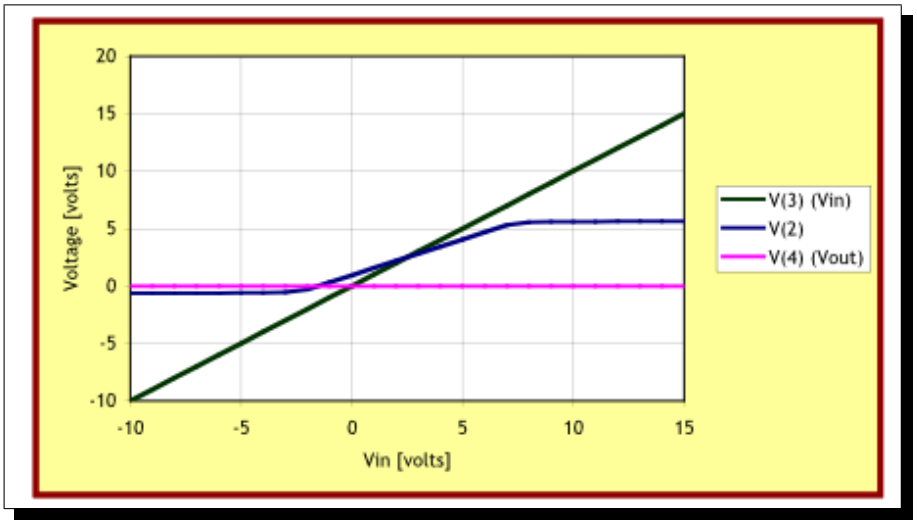

Figure 3.4. DC sweep voltages at Vin, node 2 and Vout.

```
Diode Clipper Circuit with DC sweep analysis statement
*
* Voltage Sources
VCC 1 0 5V
VIN 3 0 0V
* Analysis Command
.DC VIN -10 15 1
* Output
.PRINT DC V(3) V(2) V(4)
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
*
* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE:  DIODE
* SUBTYPE:  RECTIFIER
.MODEL D1N3940 D(
+          IS = 4E-10
+          RS = .105
+           N = 1.48
+          TT = 8E-7
+         CJO = 1.95E-11
+          VJ = .4
+           M = .38
+          EG = 1.36
+         XTI = -8
+          KF = 0
+          AF = 1
+          FC = .9
+          BV = 600
+         IBV = 1E-4)
*
.END
```


Figure 3.3. Diode clipper circuit netlist for DC sweep analysis.

3.3 Transient Analysis

This section contains an example of transient analysis in **Xyce**. In this example the DC clipper circuit of the previous section has been modified so that the input voltage source (`Vin`) is a time-dependent sinusoidal input source. The frequency of `Vin` is 1 kHz, and has an amplitude of 10 volts. For more details about transient analysis see Chapter 7.2 of this manual, or see the **Xyce** Reference Guide [3].

Example: transient analysis

To set up and run a transient analysis using the diode clipper circuit:

1. Open the diode clipper circuit netlist file file (clipper.cir) using a standard text editor (*e.g.* VI, Emacs, Notepad, *etc.*).
2. If you added DC analysis and output statements in the previous example (Figure 3.4), remove them.
3. Enter the analysis control in the netlist:

```
.TRAN 2ns 2ms
```

4. Enter the output control statement:

```
.PRINT TRAN V(3) V(2) V(4)
```

5. Modify the input voltage source (`Vin`) to generate the sinusoidal input signal:

```
VIN 3 0 SIN(0V 10V 1kHz)
```

6. At this point, the netlist should look similar to the netlist in Figure 3.5. Save the netlist file and run **Xyce** on the circuit. For example, to run serial **Xyce**:

```
> runxyce clipper.cir
```

7. Open the results file and examine (or plot) the output voltages for nodes 3 (`Vin`), 2 and 4 (`Out`). The plot in Figure 3.6 shows the output plotted as a function of time.

The modified netlist is shown in Figure 3.5, and the corresponding results in Figure 3.6.

```
Diode Clipper Circuit with transient analysis statement
*
* Voltage Sources
VCC 1 0 5V
VIN 3 0 SIN(0V 10V 1kHz)
* Analysis Command
.TRAN 2ns 2ms
* Output
.PRINT TRAN V(3) V(2) V(4)
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
*
* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE:  DIODE
* SUBTYPE:  RECTIFIER
.MODEL D1N3940 D(
+          IS = 4E-10
+          RS = .105
+           N = 1.48
+          TT = 8E-7
+         CJO = 1.95E-11
+          VJ = .4
+           M = .38
+          EG = 1.36
+         XTI = -8
+          KF = 0
+          AF = 1
+          FC = .9
+          BV = 600
+         IBV = 1E-4)
*
.END
```


Figure 3.5. Diode clipper circuit netlist for transient analysis.

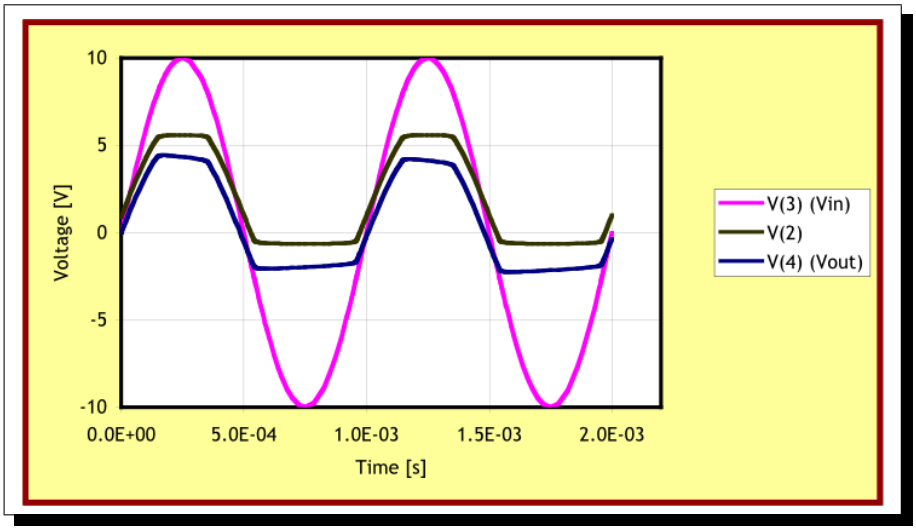

Figure 3.6. Sinusoidal input signal and clipped outputs.

4. Netlist Basics

Chapter Overview

This chapter contains introductory material on netlist syntax and usage. Sections include:

- Section 4.1 *General Overview*
- Section 4.2 *Devices Available for Simulation*
- Section 4.3 *Parameters and Expressions*

4.1 General Overview

Introduction

Using a netlist to describe a circuit for **Xyce** is the primary method for running a circuit simulation. Netlist support within **Xyce** largely conforms to that used by Berkeley SPICE 3F5 with several new options for controlling functionality unique to **Xyce**. In a netlist, the circuit is described by a set of *element lines* which define the circuit elements, their values, the circuit topology (*i.e.* the connection of the circuit elements), and a variety of control options for the simulation. The first line in the netlist file must be a title and the last line must be "`.END`". Between these two constraints, the order of the statements is irrelevant.

Nodes

Nodes and elements form the foundation for the circuit topology. Each node represents a point in the circuit that is connected to the leads of multiple elements (devices). Each lead of every element is connected to a node, and each node is connected to multiple element leads.

A node is simply a named point in the circuit. The naming of normal nodes is only known within the level of circuit hierarchy where they appear; normal nodes defined in the main circuit are not visible to subcircuits, nor are nodes defined in a subcircuit visible to the top-level circuit. Nodes can be passed into subcircuits through an argument list, and in this case subcircuits are given limited access to nodes from the upper-level circuit.

Global Nodes

For cases where a particular node is used widely throughout various subcircuits it can be more convenient to use a global node, which is referenced by the same name throughout the circuit. This is often the case for power rails such as `VDD` or `VSS`.

Global nodes start with the prefix `$G`. Examples of global node names would be: `$G_VDD` or `$G1`. There is no declaration required for nodes or global nodes. They are declared implicitly by appearing in *element lines*.

Elements

An *element line*, for which the format is determined by the specific element type, defines each circuit element instance. The general format is given by:

```
<type><name> <node information> <element information...>
```

The `<type>` must be a letter (A through Z) and the `<name>` follows immediately. For example, `RARESISTOR` specifies a device of type "R" (for "Resistor") with a name `ARESISTOR`. Nodes are separated by spaces, and additional element information required by the device is given after the node list as described in the Netlist Reference section of the **Xyce** Reference Guide [3]. **Xyce** ignores character case when reading a netlist such that `RARESISTOR` is equivalent to `raresistor`. The only exception to this case insensitivity occurs when including external files in a netlist where the filename specified in the netlist must have the same case as the actual filename.

A number field may be an integer or a floating-point value. Either one may be followed by one of the following scaling factors:

Symbol	Equivalent Value
T	10^{12}
G	10^{9}
Meg	10^{6}
K	10^{3}
mil	25.4^{-6}
m	10^{-3}
u (μ)	10^{-6}
n	10^{-9}
p	10^{-12}
f	10^{-15}

Node information is given in terms of node names, which are arbitrary character strings. The only requirement is that the ground node is named '0'. There are some restrictions on the circuit topology:

- There can be no loop of voltage sources and/or inductors.
- There can be no cut-set of current sources and/or capacitors.

In addition to these two requirements, the following additional topology constraints are highly recommended:

- Every node has a DC path to ground.
- Every node hasat least two connections (with the exception of unterminated transmission lines and MOSFET substrate nodes).

While **Xyce** can theoretically handle netlists which violate the above two constraints, such topologies are typically the result of human error in creating a netlist file and will often lead to convergence failures. See Chapter 14 of this guide for more on this topic.

The following line provides an example of an element line that defines a resistor between nodes `1` and `3` with a resistance value of $10\text{k}\Omega$.

Example: `RARESISTOR 1 3 10K`

Title, Comments and End

The first line of the netlist is the title line of the netlist. This line is treated as a comment even if it does not begin with an asterisk. It is a common mistake to forget the meaning of this first line and begin the circuit elements on the first line; doing so will probably result in a parsing error.

Example: `Test RLC Circuit`

The "`.END`" line must be the last line in the netlist.

Example: `.END`

Comments are supported in netlists and are indicated by placing an asterisk at the beginning of the comment line. They may occur anywhere in the netlist *but* they must be at the beginning of a line. **Xyce** also supports *in-line* comments. An in-line comment is designated by a semicolon and may occur on any line. Everything after the semicolon is

taken as a comment and ignored. Any line that begins with leading white space is also considered to be a comment.

Example: `* This is a netlist comment.`

Example: **WRONG:** `.DC .... * This type of in-line comment is not supported.`

Example: `.DC .... ; This type of in-line comment is supported.`

Continuation Lines

Any line that begins with a + symbol is a continuation line. Its contents are appended to those of the previous line. If the previous line or lines were comments, the continuation line is appended to the first non-comment line preceding it.

Netlist Commands

Command elements are used to describe the analysis being defined by the netlist. Examples include analysis types, initial conditions, device models and output control. The **Xyce** Reference Guide [3] contains a reference for these commands.

Example: `.PRINT TRAN V(Vout)`

Analog Devices

The analog devices supported include most of the standard circuit components normally found in circuit simulators such as SPICE 3F5, PSpice, *etc.*, plus several Sandia specific devices.

Example: `D_CR303 N_0065 0 D159700`

To find out more about analog devices see the **Xyce** Reference Guide [3].

4.2 Devices Available for Simulation

This section describes the different types of analog devices supported in **Xyce**. These include standard analog devices, sources (dependent and independent) and subcircuits. Each device description has the following information:

- A description and an example of the netlist syntax
- The corresponding model types and descriptions, where applicable
- The corresponding lists of model parameters and descriptions, where applicable
- The associated circuit diagram and model equations, as necessary

These analog devices include all of the standard circuit components needed for most analog circuits. User defined models may also be implemented using the `.MODEL` (model definition) statement and macromodels as subcircuits using the `.SUBCKT` (subcircuit) statement.

Analog Devices

Xyce supports many analog devices, including sources, subcircuits and behavioral models. The devices are classified into device types, each of which can have one or more model types. For example, the BJT device type has two model types: NPN and PNP.

The device element statements in the netlist always start with the name of the individual device instance. The first letter of the name determines the device type. The format of the following information depends on the device type and its parameters. The Device Type summary table, Table 4.1, lists all of the analog devices supported by **Xyce**. Each standard device is then described in more detail in the following sections. Except where noted, the devices are based upon those found in [6].

Table 4.1 is a summary of the analog device types and the form of their netlist formats. For a more complete description of the syntax for supported devices, see the **Xyce** Reference Guide. [3].

Device Type	Designator Letter	Typical Netlist Format
Nonlinear Dependent Source (B Source)	B	B<name> <+ node> <- node> + <I or V>={<expression>}
Capacitor	C	C<name> <+ node> <- node> [model name] <value> + [IC=<initial value>]
Diode	D	D<name> <anode node> <cathode node> + <model name> [area value]
Voltage Controlled Voltage Source	E	E<name> <+ node> <- node> <+ controlling node> + <- controlling node> <gain>
Current Controlled Current Source	F	F<name> <+ node> <- node> + <controlling V device name> <gain>
Voltage Controlled Current Source	G	G<name> <+ node> <- node> <+ controlling node> + <- controlling node> <transconductance>
Current Controlled Voltage Source	H	H<name> <+ node> <- node> + <controlling V device name> <gain>
Independent Current Source	I	I<name> <+ node> <- node> [[DC] <value>] + [transient specification]
Mutual Inductor	K	K<name> <inductor 1> [<ind. n>*] + <linear coupling or model>
Inductor	L	L<name> <+ node> <- node> [model name] <value> + [IC=<initial value>]
JFET	J	J<name> <drain node> <gate node> <source node> + <model name> [area value]
MOSFET	M	M<name> <drain node> <gate node> <source node> + <bulk/substrate node> [SOI node(s)] + <model name> [common model parameter]*
Bipolar Junction Transistor (BJT)	Q	Q<name> <collector node> <base node> + <emitter node> [substrate node] + <model name> [area value]
Resistor	R	R<name> <+ node> <- node> [model name] <value> + [L=<length>] [W=<width>]
Voltage Controlled Switch	S	S<name> <+ switch node> <- switch node> + <+ controlling node> <- controlling node> + <model name>
Transmission Line	T	T<name> <A port + node> <A port - node> + <B port + node> <B port - node> + <ideal specification>

Device Type	Designator Letter	Typical Netlist Format
Independent Voltage Source	V	V<name> <+ node> <- node> [[DC] <value>] + [transient specification]
Subcircuit	X	X<name> [node]* <subcircuit name> + [PARAMS:[<name>=<value>]*]
Current Controlled Switch	W	W<name> <+ switch node> <- switch node> + <controlling V device name> <model name>
Digital Devices	Y<name>	Y<name> [node]* <model name>
PDE Devices	YPDE	YPDE <name> [node]* <model name>
MESFET	Z	Z<name> <drain node> <gate node> <source node> + <model name> [area value]

Table 4.1: Analog Device Quick Reference.

4.3 Parameters and Expressions

In addition to explicit values, the user may use parameters and expressions to symbolize numeric values in the circuit design.

Parameters

A parameter is a symbolic name that represents a numeric value. Parameters must start with a letter or underscore. The characters after the first can be letter, underscore, or digits. Once you have defined a parameter (declared its name and given it a value) at a particular level in the circuit hierarchy, you can use it to represent circuit values at that level or any level directly beneath it in the circuit hierarchy. One way that you can use parameters is to apply the same value to multiple part instances.

How to Declare and Use Parameters

In order to use a parameter in a circuit, one must:

- define the parameter using a `.PARAM` statement within a netlist

- replace an explicit value with the parameter in the circuit

Note that **Xyce** reserves several keywords that may not be used as parameter names. These are:

- `Time`
- `Vt`
- `Temp`
- `GMIN`

While these are reserved keywords an not available for use as parameter names, only `Time` is predefined in this release of **Xyce**.

Example: Declaring a parameter

1. Locate the level in the circuit hierarchy at which the `.PARAM` statement declaring a parameter will be placed. (Note: a parameter that can be used anywhere in the netlist can be declared by placing the `.PARAM` statement at the top-most level of the circuit.)
2. Name the parameter and give it a value. The value can be numeric or given by an expression:

```
.SUBCKT subckt1 n1 n2 n3
.PARAM res = 100
*
* other netlist statements here
*
.ENDS
```

3. Note: the parameter *res* can be used anywhere within the subcircuit subckt1 including subcircuits defined within it, but cannot be used outside of subckt1.

Example: Using a parameter in the circuit

1. Find the numeric value that is to be replaced by a parameter: a device instance parameter value, model parameter value, *etc.* The value being replaced must be accessible with the current hierarchy level.

2. Replace the numeric value with the parameter name contained within braces ({}) as in:

```
R1 1 2 {res}
```

Limitations on parameter definitions

There is considerable flexibility in the use of parameters, as described in section 6, analog behavioral modelling. Parameters can be set to expressions containing other parameters, and can be passed down the heirarchy into subcircuits. Fundamentally, however, parameters are constants that are evaluated at the beginning of a run. All terms in the expression defining the parameter must therefore be constants known at the beginning of the run. It is not legal to use any time-dependent expressions in parameter declarations (either by including voltage nodes or currents, or by including reference to the variable TIME).

If a parameter is defined within a given scope then it can be used in any expression within that scope. The only limitation on ordering is for the use of a parameter in an expression that defines the value of another parameter. In that case, all parameters used in the expression must be defined before being used to define another parameter. So,in the following example:

```
R1  1  0  {B+C} ; OK because the expression is not used to define a param
.PARAM A=3
.PARAM B={A+1}  ; OK because A is defined above
.PARAM D={C+2}  ; Illegal because C is not yet known
.PARAM C=2
```

Global Parameters

A normal parameter that is defined at the main circuit level will have global scope. Such parameter suffer from limitations that (1) they are constant during the simulation, and (2) the parameter may redefined within a subcircuit, which would change the value in the subcircuit and below. Global parameters address these limitations.

A global parameter differs from a normal parameter in that is can only be defined at the main circuit level, and it is allowed to change during a simulation. It acts like a variable rather than a constant during the simulation. An example of some global parameter usages are:

```
.param dTdt=100
.global_param T={27+dTdt*time}
R1  1  2  RMOD TEMP={T}

or . .

.global_param T=27
R1  1  2  RMOD TEMP={T}
C1  1  2  CMOD TEMP={T}
.step T 20 50 10
```

In these examples, T is used to represent an environmental variable that changes.

Note that normal parameters may be used in expressions that define global parameters, but the the opposite is not allowed.

Expressions

In **Xyce**, an expression is a mathematical relationship that may be used any place one would use a number (numeric or boolean). Except in the case of expressions used in analog behavioral modeling sources (see Chapter 6) **Xyce** evaluates the expression to a value when it reads in the circuit netlist, not each time its value is needed. It is therefore necessary that all terms in an expression be known at the beginning of the run.

To use an expression in a circuit netlist:

1. Locate the value to be replaced (component, model parameter, *etc.*).
2. Substitute the value with an expression utilizing the $\{\}$ syntax:

   $\{expression\}$

   where *expression* can contain any of the following:

   - available operators from those in Table 4.2
   - included functions from those in Table 4.3, Table 4.4, and Table 4.5
   - user-defined functions
   - user-defined parameters that are within scope
   - literal operands

The braces ({}) instruct **Xyce** to evaluate the expression and use the resulting value.

Additional time-dependent constructs are available in expressions used in analog behavioral modeling sources (see Chapter 6).

Example: Using an expression

Scaling the DC voltage of a $12V$ independent voltage source, designated `VF`, by some factor can be accomplished by the following netlist statements (in this example the factor is 1.5):

```
.PARAM FACTORV=1.5
VF 3 4 {FACTORV*12}
```

Xyce will evaluate the expression to $12 * 1.5$ or 18 volts.

[1] Logical and relational operators are used only with the `IF()` function.

Class of operator	Operator	Meaning
arithmetic	+	addition or string concatenation
	-	subtraction
	*	multiplication
	/	division
	**	exponentiation
logical[1]	~	unary NOT
	|	boolean OR
	^	boolean XOR
	&	boolean AND
relational	==	equality
	!=	non-equality
	>	greater-than
	>=	greater-than or equal
	<	less-than
	<=	less-than or equal

Table 4.2. Expression operators

Function	Meaning	Explanation
ABS(x)	$\|x\|$	absolute value of x
DDT(x)	$\frac{d}{dt}x(t)$	time derivative of x
DDX(f(x),x)	$\frac{\partial}{\partial x}f(x)$	partial derivative of $f(x)$ with respect to x
IF(t,x,y)	x if t is true, y otherwise	t is an expression using the relational operators in Table 4.2
LIMIT(x,y,z)	y if $y < x$ x if $y < x < z$ z if $x < z$	x limited to range y to z
M(x)	$\|x\|$	absolute value of x
MIN(x,y)	$\min(x, y)$	minimum of x and y
MAX(x,y)	$\max(x, y)$	maximum of x and y
PWR(x,y)	x^y	x raised to y power
PWRS(x,y)	x^y if $x > 0$ 0 if $x = 0$ $-(-x)^y$ if $x < 0$	sign corrected x raised to y power
RAND()	$0 < result < 1$	random constant number between 0 and 1
SDT(x)	$\int x(t)dt$	time integral of x
SGN(x)	+1 if $x > 0$ 0 if $x = 0$ -1 if $x < 0$	sign value of x
STP(x)	1 if $x > 0$ 0 otherwise	step function
SQRT(x)	$\sqrt{x}$	square root of x
TABLE(x,y,z,*)	$f(x)$ where $f(y) = z$	piecewise linear interpolation, multiple (y,z) pairs can be specified
URAMP(x)	x if $x > 0$ 0 otherwise	ramp function

Table 4.3. Arithmetic Functions in Expressions

Function	Meaning	Explanation
ACOS(x)	$\arccos(x)$	result in radians
ACOSH(x)	$\cosh^{-1}(x)$	hyperbolic arccosine of x
ARCTAN(x)	$\arctan(x)$	result in radians
ASIN(x)	$\arcsin(x)$	result in radians
ASINH(x)	$\sinh^{-1}(x)$	hyperbolic arcsine of x
ATAN(x)	$\arctan(x)$	result in radians
ATANH(x)	$\tanh^{-1}(x)$	hyperbolic arctangent of x
ATAN2(x,y)	$\arctan(x/y)$	result in radians
COS(x)	$\cos(x)$	x in radians
COSH(x)	$\cosh(x)$	hyperbolic cosine of x
EXP(x)	e^x	e to the x power
LN(x)	$\ln(x)$	log base e
LOG(x)	$\log(x)$	log base 10
LOG10(x)	$\log(x)$	log base 10
SIN(x)	$\sin(x)$	x in radians
SINH(x)	$\sinh(x)$	hyperbolic sine of x
TAN(x)	$\tan(x)$	x in radians
TANH(x)	$\tanh(x)$	hyperbolic tangent of x

Table 4.4. Exponential, Logarithmic, and Trigonometric Functions in Expressions

Function	Meaning	Explanation
SPICE_EXP(V1,V2, TD1,TAU1,TD2,TAU2)		SPICE style exponential (function of time) V1 = initial value V2 = pulsed value TD1 = rise delay time TAU1 = rise time constant TD2 = fall delay time TAU2 = fall time constant
SPICE_PULSE(V1,V2, TD,TR,TF,PW,PER)		SPICE style pulse (function of time) V1 = initial value V2 = pulsed value TD = delay TR = rise time TF = fall time PW = pulse width PER = period
SPICE_SFFM(V0,VA, FC,MDI,FS)		SPICE style single frequency FM (function of time) V0 = offset VA = amplitude FC = carrier frequency MDI = modulation index FS = signal frequency
SPICE_SIN(V0,VA FREQ,TD,THETA)		SPICE style sine wave (function of time) V0 = offset VA = amplitude FREQ = frequency (hz) TD = delay THETA = damping factor

Table 4.5. SPICE Compatibility Functions in Expressions

5. Working with .MODEL Statements and Subcircuit Models

Chapter Overview

This chapter contains model ideas and a summary of the ways to create and modify models. Sections include:

- Section 5.1, *Definition of a Model*
- Section 5.2, *Model Organization*

5.1 Definition of a Model

A model describes the electrical performance of a *part*, such as a specific vendor's version of a 2N2222 transistor. To simulate a part requires specification of *simulation properties*. These properties *define* the model of the part.

Depending on the given device type and the requirements of the circuit design, a model is specified using a model parameter set, a subcircuit netlist, or both.

In general, *model parameter sets* define the parameters used in ideal models of specific device types, and *subcircuit netlists* allow the user to combine ideal device models to simulate more complex effects. For example, one could simulate a bipolar transistor using the Xyce BJT device by specifying model parameters extracted to fit the simulation behavior to the behavior of the part used, or one could develop a subcircuit macro-model of a capacitor that adds effects like lead inductance and resistance to the basic capacitor device.

Both methods of defining a model use a netlist format, with precise syntax rules as described below.

Defining models using model parameters

Xyce currently has no built-in part models. However, models can be defined for a device by changing some or all of the *model parameters* from their defaults via the `.MODEL` statement. For example:

```
M5 3 2 1 0 MLOAD1
.MODEL MLOAD1 NMOS (LEVEL=3 VTO=0.5 CJ=0.025pF)
```

This example defines a MOSFET device `M5` that is an instance of a part described by the model parameter set `MLOAD1`. The `MLOAD1` parameter set is defined in the `.MODEL` statement.

Most device types in **Xyce** support some form of model parameters. Consult the **Xyce** Reference Guide [3] for the model parameters supported by each device type.

Defining models using subcircuit netlists

In **Xyce**, models may also be defined using the *subcircuit syntax:* `.SUBCKT`/`.ENDS`. This syntax includes:

- *netlists* to define the configuration and function of the part.
- *variable input parameters* to modify the model.

See Figure 5.1 for an example.

```
****other devices
X5 5  6  7  8 l3dsc1 PARAMS: ScaleFac=2.0
X6 9 10 11 12 l3dsc1
****more netlist commands

*** SUBCIRCUIT: l3dsc1
*** Parasitic Model: microstrip
*** Only one segment
.SUBCKT l3dsc1 1 3 2 4 PARAMS: ScaleFac=1.0
C01 1 0 4.540e-12
RG01 1 0 7.816e+03
L1 1 5 3.718e-08
R1 5 2 4.300e-01
C1 2 0 4.540e-12
RG1 2 0 7.816e+03
C02 3 0 4.540e-12
RG02 3 0 7.816e+03
L2 3 6 3.668e-08
R2 6 4 4.184e-01
C2 4 0 4.540e-12
RG2 4 0 7.816e+03
CM012 1 3 5.288e-13
KM12 L1 L2 2.229e-01
CM12 2 4 {5.288e-13*ScaleFac}
.ENDS
```

Figure 5.1. Example subcircuit model.

In this example, a subcircuit model called `l3dsc1` implementing one part of a microstrip transmission line is defined between the `.SUBCKT`/`.ENDS` lines, and two different instances of the subcircuit are used in the `X` lines. This somewhat artificial example shows how input parameters are used; the last capacitor in the subcircuit is scaled by the input parameter `ScaleFac`. If input parameters are not specified on the `X` line (as in the case of device `X6`), then the default values specified on the `.SUBCKT` line are used. Non-default values are specified on the `X` line using the `PARAMS:` keyword. For precise syntax consult the **Xyce** Reference Guide [3].

Subcircuit Hierarchy

Xyce supports the definition of subcircuits within other subcircuits. Each subcircuit definition introduces a new level in the circuit hierarchy with the top level being the main circuit. If a second level is defined, it is composed of the subcircuits in the main circuit and each subsequent level is composed of the subcircuits contained in the previous level. A subcircuit may also contain other definitions such as models via the `.MODEL` statement, parameters via the `.PARAM` statement, and functions via the `.FUNC` statement.

In this context, the subcircuit defines the "scope" for the definitions it contains. That is, *the definitions contained within a subcircuit can be used within that subcircuit and/or within any subcircuit it contains*. Any definitions occurring in the main circuit have global scope and can be used anywhere in the circuit. A name, such as a model, parameter, function or subcircuit name, occurring in a definition at one level of a circuit hierarchy can be redefined at any lower level contained directly by the subcircuit. In this case, the new definition applies at the given level and those below.

In the following example, the model named `MOD1` can be used in subcircuits `SUB1` and `SUB2` but not in the subcircuit `SUB3`. The parameter `P1` has a value of 10 in subcircuit `SUB1` and a value of 20 in subcircuit `SUB2`.

```
.SUBCKT SUB1 1 2 3 4
.MODEL MOD1 NMOS(LEVEL=2)
.PARAM P1=10
*
* subcircuit devices omitted for brevity
*
.SUBCKT SUB2 1 3 2 4
.PARAM P1=20
*
* subcircuit devices omitted for brevity
*
.ENDS
.ENDS

.SUBCKT SUB3 1 2 3 4
*
* subcircuit devices omitted for brevity
*
.ENDS
```

Figure 5.2. Example subcircuit model.

5.2 Model Organization

While it is always possible to make a self-contained netlist in which all models for all parts are included along with the circuit definition, it is often more convenient to organize frequently-used models into separate model libraries. **Xyce** provides a very simple mechanism that allows this organization. Models are simply collected into model library files, and then accessed by netlists as needed by insertion of an `.INCLUDE` directive. This section describes the process in detail.

Model libraries

Device model and subcircuit definitions may be organized into model libraries. These libraries are text files (similar to netlist files) that have one or more model definitions. Many users choose to give model library files names that end with `.lib`, but the file may be named using any convention the user chooses.

In general, most users create model libraries files that typically include similar model types. In these files, the *header comments* describe the models therein.

Model library configuration

In **Xyce**, model libraries are used by inserting a `.INCLUDE` statement into a netlist. Once a file is included, its contents are available to the netlist just as if the entire contents had been inserted directly into the netlist.

As an example, one might create the following model library file called `bjtmodels.lib`, containing `.MODEL` statements for common types of bipolar junction transistors:

```
*bjtmodels.lib
* Bipolar transistor models
.MODEL Q2N2222 NPN (Is=14.34f Xti=3 Eg=1.11 Vaf=74.03 Bf=5 Ne=1.307
+  Ise=14.34f Ikf=.2847 Xtb=1.5 Br=6.092 Nc=2 Isc=0 Ikr=0 Rc=1
+  Cjc=7.306p Mjc=.3416 Vjc=.75 Fc=.5 Cje=22.01p Mje=.377 Vje=.75
+  Tr=46.91n Tf=411.1p Itf=.6 Vtf=1.7 Xtf=3 Rb=10)

.MODEL 2N3700 NPN (IS=17.2E-15 BF=100)

.MODEL 2N2907A PNP (IS=1.E-12 BF=100)
```

The models Q2N2222, 2N3700 and 2N2907A could then be used in a netlist by including the bjtmodels.lib file.

```
.INCLUDE "bjtmodels.lib"
Q1 1 2 3 Q2N2222
Q2 5 6 7 2N3700
Q3 8 9 10 2N2907A
*other netlist entries
.END
```

Because the contents of an included file are simply inserted into the netlist at the point where the .INCLUDE statement appears, the scoping rules for .INCLUDE statements is the same as for other types of definitions as outlined in the preceding section. Note that the path to the library file is assumed to be relative to the execution directory, but absolute pathnames are permissible. Note also that the entire file name, including its "extension" must be specified. There is no assumed default extension.

5.3 Model Interpolation

Modelling of thermal effects is handled in a variety of ways, depending on the device type. For simple devices this is by linear and quadratic correction factors. For the diodes this is by self contained temperature correction equations, which have been verified to be accurate. For complex semiconductor devices the only accurate method is through interpolation of models measured at multiple nominal temperatures (TNOM).

Interpolation of models is accessed through the model parameter TEMPMODEL in the models that support this capability. In the netlist, a base model is specified, and is followed by multiple models at other temperatures. This is best shown by example:

```
Jtest 1a 2a 3 SA2108 TEMP= 40
*
.MODEL SA2108 PJF ( TEMPMODEL=QUADRATIC TNOM = -55
+ LEVEL=2 BETA = 0.00365 VTO = -1.9360 PB = 0.304
+ LAMBDA = 0.00286 DELTA = 0.2540 THETA = 0.0
+ IS = 1.393E-10 RD = 0.0 RS = 1e-3)

.MODEL SA2108 PJF ( TEMPMODEL=QUADRATIC TNOM = 27
+ LEVEL=2 BETA= 0.003130 VTO = -1.9966 PB = 1.046
+ LAMBDA = 0.00401 DELTA = 0.578; THETA = 0;
+ IS = 1.393E-10          RS = 1e-3)
*
.MODEL SA2108 PJF ( TEMPMODEL=QUADRATIC TNOM = 90
+ LEVEL=2 BETA = 0.002770 VTO = -2.0350 PB = 1.507
+ LAMBDA = 0.00528 DELTA = 0.630 THETA = 0.0
+ IS = 1.393E-10          RS = 5.66)
```

Note that the model names are all identical for three .MODEL lines, and they all specify TEMPMODEL=QUADRATIC, but with different TNOM. For parameters that appear in all three .MODEL lines, the value of the parameter will be interpolated using the TEMP= value in the device line, which in this example is 40 degree C, in the first line. For parameters that are not interpolated, like RD, it is not necessary to include these in the second and third .MODEL lines.

Currently, the only arguments for TEMPMODEL are QUADRATIC and PWL (piecewise linear). The quadratic method also has a limiting feature that does not allow the parameter value to go outside the range of values specified in the .MODEL lines. For example, the

value of RS in the example would take on negative values for most of the interval between -55 and 27 since the value at 90 is very high. This truncation is necessary since parameters can easily take on values that will cause failure of Xyce, like the negative resistance of RS in this example.

For certain parameters, interpolation is done on the the log of the parameter rather than the parameter itself. So far, this is only applicable to the BJT parameters IS and ISE. This gives excellent interpolation of these parameters, which vary over many orders of magnitude, and have this type of dependence on temperature.

6. Analog Behavioral Modeling

Chapter Overview

This chapter contains a description of analog behavioral modeling in **Xyce**. Sections include:

- Section 6.1,*Overview of Analog Behavioral Modeling*
- Section 6.2, *Specifying ABM Devices*
- Section 6.3, *Guidance for ABM Use*

6.1 Overview of Analog Behavioral Modeling

The analog behavioral modeling capability of **Xyce** provides for flexible descriptions of electronic components in terms of a transfer function or lookup table. In other words, a mathematical relationship is used to model a circuit segment removing the need for component by component design.

The primary device used for analog behavioral modeling in **Xyce** is the `B` device, or nonlinear dependent source. A `B` device can serve as a voltage or current source, and by using expressions dependent on voltages and currents elsewhere in the circuit the user can produce any desired behavior.

6.2 Specifying ABM Devices

ABM devices (`B` devices) are specified in a netlist the same way as other devices. Customizing the operational behavior of the device is achieved by defining an ABM expression describing how inputs are transformed into outputs.

For example, the pair of lines below would provide exactly the same behavior as a 10K resistor between nodes 1 and 2. It is written to be a current source with the current specified using Ohm's law and the constant resistance.

```
.PARAM Res1=10K
Blinearres 1 2 I={(V(2)-V(1))/Res1}
```

A nonlinear resistor could be specified similarly:

```
.PARAM R1=0.15
.PARAM R2=6
.PARAM E2 = {2*E1}
.PARAM delr = {R1-R0}
.PARAM k1 = {1/E1**2}
.PARAM r2 = {R0+sqrt(2)*delr}
```

```
.FUNC Rreg1(a,b,c,d) {a +(b-a)*c/d}
.Func Rreg2(a,b,c,d,f) {a+sqrt(2-b*(2*c-d)**2)*f}

Bnlr 4 2 V = {I(Vmon) * IF(
+ V(101) < E1, Rreg1(R0,R1,V(101),E1),
+ IF(
+ V(101) < E2, Rreg2(R0,k1,E1,V(101),delr), R2
+ )
+ )}
```

In this example, `Bnlr` provides a voltage between nodes 4 and 2, and the voltage is determined using Ohm's law with a resistance that is a function of the voltage on node 101 and a number of parameters. These two examples demonstrate how the `B` source can be used either as a voltage source (by specifying `V={expression}`) or as a current source (with `I={expression}`).

Note that unlike expressions used in parameters or function declarations, expressions in the nonlinear dependent source may contain voltages and currents from other parts of the circuit, or even explicit time-dependent functions. These expressions are evaluated every time the current or voltage through the source are needed.

Additional constructs for use in ABM expressions

ABM expressions follow the same rules as other expressions in a netlist with the additional ability to specify signals (node voltages and voltage source currents) and explicitly time-dependent functions in the expression. In ABM expressions, refer to signals by name. **Xyce** recognizes the following constructs in ABM expressions:

- `V(<node name>)`
- `V(<node name>,<node name>)`
- `I(<voltage source name>)`
- `I(<two terminal device name>)`
- `I<lead deignator>(three or more terminal device name>)`
- The variable `TIME`
- Lookup tables

In a hierarchical circuit (a circuit with possibly nested levels of subcircuits), voltage source names that appear in an ABM expression must be the name of a voltage source in the same subcircuit as the ABM device. Similarly, node names in an ABM expression must be the node names of one or more devices in the same subcircuit as the ABM device.

Examples of Analog Behavioral Modeling

A variety of examples of legal usage of analog behavioral modelling is probably the most effective means of demonstrating what is allowed. The following shows the range of simple items allowed in such expressions:

```
B1  1  0  I={V(2,3) + I(R4)}
R4  2  0  10K
Em  3  2  VALUE={PAR3+1000*time}
B2  2  0  V={I(Em)}
M3  8  6  0  NMOD
B3  6  4  V={ID(M3)+V(2)}
```

The range of items that can be used in the current and voltage parameters of a B (or E, F, G, or H) source is far greater than what is allowed for expressions in other contexts. In particular, the use of solution values (V(*), V(*,*), I(V*)), and lead currents (I(*) and I*(*)) are prohibited in all other expressions because they lead to unstable behavior if used elsewhere. Time dependent expressions are allowed for some device parameters, but this feature should be used with caution as the behavior of the device cannot be guaranteed to be correct when device parameters are not constant throughout the run.

In addition to these simple items, lookup tables provide a means of specifying a piecewise-linear function in an expression. A table expression is specified with the keyword `TABLE` followed by an expression that is evaluated as the independent variable of the piecewise linear function, followed by a list of pairs of independent variable/dependent variable values. For example

Example: B1 1 0 V={TABLE {time} = (0, 0) (1, 2) (2, 4) (3, 6)}

An equivalent example uses the table function, which has a simpler syntax, but which may be hard to read for long tables:

Example: B1 1 0 V={TABLE(time, 0, 0, 1, 2, 2, 4, 3, 6)}

These examples will produce a voltage source whose voltage is a simple linear function of time. At $t = 0$ the voltage is 0 volts, at time $t = 1s$ the voltage is 2 volts, and at times in between the voltage is determined by linear interpolation.

The independent variable of the table source does not have to be a simple expression:

Example: B1 1 0 V={TABLE {V(5)-V(3)/4+I(V6)*Res1} = (0, 0) (1, 2) (2, 4) (3, 6)}

Alternate behavioral modeling sources

In addition to the primary nonlinear dependent source, the `B` source, **Xyce** also supports the PSpice extensions to the standard Spice voltage- and current-controlled sources, the `E`, `F`, `G` and `H` sources. These sources are provided for PSpice compatibility, and are converted internally into equivalent `B` sources. See the Netlist Reference chapter of the **Xyce** Reference Guide [3] for the syntax of these compatibility devices.

6.3 Guidance for ABM Use

ABM devices add equations to the problem

It is important that the user keep in mind that Xyce solves a complex nonlinear set of equations at each time step, that this system of equations is solved iteratively to obtain a converged solution, and that specifying an ABM device in a Xyce netlist adds one or more equations to the nonlinear problem that Xyce has to solve. These ABM equations are such that when the nonlinear problem is converged the voltage drop across the ABM device (or current through the device) satisfies the expression given. But during the course of the iterative solution the unconverged values of nodal voltages and currents are not solutions to the system of equations, do not obey basic physical requirements of circuit solutions (Kirchhoff's laws), and might in fact take on values that are "unphysical" or of the wrong sign. Only at the end of the nonlinear iterative solution are the solution variables consistent, legal values.

Be sure that all expressions used in ABM devices are valid for any possible value of their independent variables

While ABM devices look temptingly like calculators, it is potentially dangerous to use them as such. In the previous subsection, it was stated that during the nonlinear solution of each timestep's equations the nodal voltages and currents are usually not solutions to the full set of equations, and often violate Kirchhoff's laws. Only at the end of the nonlinear solution are all the constraints on voltages and currents satisfied. This has some important consequences to the user of ABM devices.

All expressions involving nodal voltages and currents used in ABM devices should be valid for any possible value they might see — even those that appear to be physically meaningless and which a knowledgeable user might never expect to see in the real circuit. This is particularly important when using square roots or exponentiating to a fractional power. For example, consider the apparently trivial netlist fragment below:

```
*...other parts of more complex circuit deleted...
* potentially bad usage of ABM device
Vexample 1 0 5V
d1 1 0 diode_model
B1 2 0 V={sqrt(v(1))}
r1 2 0 10k
*...other parts of more complex circuit deleted...
```

This is a contrived example, but demonstrates a potentially dangerous usage. It is assumed, because node 1 is connected to a 5V DC source, that the argument of the square root function is always positive. However it could be the case that during the nonlinear solution of the full circuit that an unconverged value of node 1 might be negative. Tracking down mistakes like this can be difficult, as what tends to happen is that on most platforms B1 results in a "Not a Number" value for the nodal voltage of node 2, but the program doesn't crash. This frequently shows up as inexplicable "Timestep too small" errors.

Such things can be avoided by protecting the arguments of functions with limited domain, but care needs to be taken when doing this. One obvious way to protect the circuit fragment above would be to take the absolute value of V(1) before calling the sqrt function:

```
*...other parts of more complex circuit deleted...
* safer usage of ABM device
```

```
Vexample 1 0 5V
d1 1 0 diode_model
B1 2 0 V={sqrt(abs(v(1)))}
r1 2 0 10k
*...other parts of more complex circuit deleted...
```

There are numerous other ways to protect the square root function from negative arguments, such as by using the maximum of zero and V(1). Some alternatives might be more appropriate than others in different contexts.

Note, though, that it would be a mistake to attempt the absolute value like this:

```
*...other parts of more complex circuit deleted...
* really bad misuse of ABM device
Vexample 1 0 5V
d1 1 0 diode_model
B2 3 0 V={abs(v(1))} ; watch out!
B1 2 0 V={sqrt(v(3))}; just as bad as first example!
r1 2 0 10k
*...other parts of more complex circuit deleted...
```

There are two things wrong with this example — first, node 3 is floating and this alone could lead to convergence problems. Second, by adding the second ABM device one has merely created an equation whose solution is that node 3 contains the absolute value of the voltage on node 1, but until convergence is reached it is not guaranteed that node 3 will be precisely the absolute value of V(3), nor is it even guaranteed that node 3 will have a positive voltage. Only at convergence do nodes have the values that are solutions to the set of equations created by the netlist.

ABM devices should not be used purely for output post-processing

Users frequently use ABM devices to provide output post-processing. For example, if a user was interested in the absolute value, or the log of an output voltage, that user might create a ABM circuit element that calculated the desired output value.

Using ABM sources in this manner is a bad practice. By creating a circuit element whose only purpose is post-processing, **Xyce** is forced to include this bogus device in the circuit,

and the corresponding nonlinear solve. Often, this causes unnecessary solver problems. If post-processing is the goal, it is much better to use expressions directly on the .PRINT line. This is a supported capability as of **Xyce** Release 2.1.

```
* Bad example
B1 test1 0 V = {(abs(I(VMON)))*1.0e-10}
VIN 1 0 DC 5V
R1 1 2 2K
D1 3 0 DMOD
VMON 2 3 0
.MODEL DMOD D (IS=100FA)
.DC VIN 5 5 1
.PRINT DC I(VMON) V(3) V(test1)
```

An example of a "bad use" of ABM sources can be found the above code fragment. The source B1, is only in the circuit to provide a post-processing output. It doesn't play a functional role in the circuit, but **Xyce** would still be forced to include it in the problem it is attempting to solve. A much better solution is to get rid of B1, and replace it with an expression in the .PRINT line. A better solution to the above problem is given here:

```
* Good example
VIN 1 0 DC 5V
R1 1 2 2K
D1 3 0 DMOD
VMON 2 3 0
.MODEL DMOD D (IS=100FA)
.DC VIN 5 5 1
.PRINT DC I(VMON) V(3)  {(abs(I(VMON)))*1.0e-10}
```

For a more detailed explanation of how to use expressions in the .PRINT line, see section 10.1, or the **Xyce** Reference Guide [3].

7. Analysis Types

Chapter Overview

This chapter contains a description of the different analysis types available in **Xyce**. It includes the following sections:

- Section 7.1, *Introduction*
- Section 7.2, *Transient Analysis*
- Section 7.3, *DC Analysis*
- Section 7.4, *STEP Parametric Analysis*

7.1 Introduction

Several simulation analysis options are supported within **Xyce**. For basic analysis, **Xyce** currently supports DC and transient analysis; AC analysis is intended to be supported in a future release. STEP parametric analysis, which applies an outer parameter loop to either DC or transient analysis is also available.

7.2 Transient Analysis

The transient response analysis simulates the response of the circuit from `TIME=0` to a specified time. Throughout a transient analysis, any or all of the independent sources may have time-dependent values.

In **Xyce** (and most other circuit simulators), the transient analysis begins by performing its own bias point calculation at the beginning of the run, using the same method as used for DC sweep. This is required to set the initial conditions for the transient solution as the initial values of the sources may differ from the their DC values.

.TRAN Statement

To run a transient simulation, the circuit netlist file must contain a `.TRAN` command.

Example:

```
.TRAN 100us 300ms
.TRAN 100p 12.05u 9.95u
```

For a detailed explanation of the `.TRAN` statement, see the **Xyce** Reference Guide [3]). In addition to a `.TRAN` statement, the netlist must contain one of the following:

- an independent, transient source (see Table 7.1),
- an initial condition on a reactive element, or
- a time-dependent behavioral modeling source (see Chapter 6)

Defining a Time-Dependent (transient) Source

Overview of Source Elements

Source elements, either voltage or current, are entered in the netlist file as described in the **Xyce** Reference Guide [3]. Table 7.1 list the time-dependent sources available in **Xyce** for either voltage or current. For voltage sources, the name is preceded by the letter `V` while current sources are preceded by the letter `I`.

Source Element Name	Description
EXP	Exponential Waveform
PULSE	Pulse Waveform
PWL	Piecewise Linear Waveform
SFFM	Frequency-modulated Waveform
SIN	Sinusoidal Waveform

Table 7.1. Summary of time-dependent sources supported by **Xyce**.

To use one of these time-dependent or transient sources, the user must place the source element line in the netlist and characterize the transient behavior using the appropriate parameters. Each transient source element has a separate set of parameters dependent on its transient behavior. In this way, the user can create analog sources which produce sine wave, square pulse, exponential pulse, single-frequency FM, and piecewise linear waveforms.

Defining Transient Sources

To define a transient source:

- Select one of the supported sources: independent voltage or current source.
- Choose a transient source type from Table 7.1.
- Provide the transient parameters (see the **Xyce** Reference Guide [3]) to fully define the source.

Below is an example of an independent sinusoidal voltage source in a circuit netlist. It creates a voltage source between nodes 1 and 5 that oscillates sinusoidally between -5V and +5V with a frequency of 50 KHz.

Example: `Vexample 1 5 SIN(-5V 5V 50KHz)`

Transient Calculation Time Steps

During the simulation, **Xyce** uses a calculation time step that is continuously adjusted for accuracy and efficiency (see [7] and [8]). During periods of circuit idleness the calculation time step is increased, and during dynamic portions of the waveform it is decreased. The maximum internal step size can be controlled by specifying the step ceiling value in the `.TRAN` command (see the **Xyce** Reference Guide [3]).

The internal calculation time steps used might not be consistent with the output time steps requested by the user. By default **Xyce** outputs solution results at every time step it calculates. If the user selects output timesteps via the `.OPTIONS OUTPUT` statement (see Chapter 10) then **Xyce** will output results for the closest time step that follows the time requested by the user. There is currently no mechanism for forcing **Xyce** to output at precise user-specified times.

Transient Time Step Selection Advice

There are two basic familes of step-size selection in Xyce. The first is Local Truncation Error (LTE) based and the second is non-LTE based. There are two options that apply to both and they directly relate to how step-sizes are chosen right after breakpoints and between breakpoints. The option MINTIMESTEPSBP specifies the minimum number of time-steps between breakpoints and effectively enforces a local maximum time-step. The option RESTARTSTEPSCALE determines the size of the first step coming out of a breakpoint. The step-size is calculated by multiplying RESTARTSTEPSCALE by the distance between the two breakpoints.

Example:

```
.OPTIONS TIMEINT MINTIMESTEPSBP=100 RESTARTSTEPSCALE=0.001
```

Local Trunction Error (LTE) Strategy

Both the BDF15 integrator and the Trapezoid integrator use the LTE based strategy by default. The strategy is to specify appropriate relative and absolute error tolerances. These are specified using .options timeint abstol=1.0e-6 reltol=1.0e-2. These are the default values.

Example:

```
.OPTIONS TIMEINT ERROPTION=0 RELTOL=1e-1 ABSTOL=1e-5
```

One feature of the Trapezoid integrator is that there is no numerical dissipation introduced by the algorithm. This means that strong ringing will occur when discontinuities are introduced by sources or models. This ringing is entirely artificially introduced by the numerical algorithm. This can result in a large local truncation error estimate ultimately leading to a time-step too small error. In this case, using a Non LTE strategy may help.

Non LTE Strategy

The Non LTE strategy used in Xyce is based on success of the nonlinear solve. This strategy is enabled by setting ERROPTION=1. The behavior of this setting is slightly different for the BDF15 integrator and the Trapezoid integrator. Since the step-size selection is based only upon nonlinear iteration statistics, and not accuracy, it is highly suggested that DELMAX be specified in a circuit specific way.

For the BDF15 integrator, if the nonlinear solver converges then the step-size is doubled. On the other hand, if the nonlinear solver fails to converge, then the step-size is cut by one eighth.

Example:

```
.OPTIONS TIMEINT ERROPTION=1 DELMAX=1.0e-4
```

For the Trapezoid integrator, the options are slightly more refined. If the number of nonlinear iterations is below NLMIN, then the step-size is doubled. If the number of nonlinear iterations is above NLMAX then the step-size is cut by one eighth. In between, the step-size is left alone.

Example:

```
.OPTIONS TIMEINT METHOD=7 ERROPTION=1 NLMIN=3 NLMAX=8 DELMAX=1.0e-4
```

By default time-steps are not rejected unless the nonlinear solver fails to converge. This is the opposite of the default mode in LTE based step-selection. This can be enabled with TIMESTEPSREVERAL=1.

Example:

```
.OPTIONS TIMEINT METHOD=7 ERROPTION=1 DELMAX=1.0e-4 TIMESTEPSREVERAL=1
```

Checkpointing and Restarting

The `.OPTIONS RESTART` command (in the netlist) is used to control all checkpoint output and restarting. Checkpointing and associated restart can be extremely useful for long simulations. In essence, **Xyce** allows the user to save the state of the simulation during a run (at intervals the user specifies) (*checkpointing*). This checkpoint data can then be read in to *restart* the simulation from any of the saved (*checkpointed*) time points.

Checkpointing Command Format

- `.OPTIONS RESTART PACK=<0|1> JOB=<job name> [INITIAL_INTERVAL=<interval> [<t0> <i0> [<t1> <i1>...]]]`

  `PACK=<0|1>` indicates whether the restart data files will contain byte packed data(1) or not(0). `JOB=<job name>` identifies the prefix for restart files. The actual restart files will be the job name appended with the current simulation time (e.g. `name1e-05` for `JOB=name` and simulation time 1e-05 seconds). Furthermore, the `INITIAL_INTERVAL=<interval>` identifies the initial interval time used for restart output. The `<tx ix>` intervals identify times (`tx`) at which the output interval (`ix`) will change. This functionality is identical to that described for the `.OPTIONS OUTPUT` command (see Section 10.1).

- Example - generate checkpoints at every time step (default):

  ```
  .OPTIONS RESTART JOB=checkpt
  ```

- Example - generate checkpoints every 0.1 μs:

  ```
  .OPTIONS RESTART JOB=checkpt INITIAL_INTERVAL=0.1us
  ```

- Example - generate unpacked checkpoints every 0.1 μs:

```
.OPTIONS RESTART PACK=0 JOB=checkpt INITIAL_INTERVAL=0.1us
```

- Example - Initial interval of 0.1 μs, at 1 μs in the simulation, change to interval of 0.5 μs, and at 10 μs change to an interval of 0.1 μs:

```
.OPTIONS RESTART JOB=checkpt INITIAL_INTERVAL=0.1us 1us 0.5us
+ 10us 0.1us
```

Restarting Command Format

- `.OPTIONS RESTART <FILE=<filename> | JOB=<job name> START_TIME=<time>>`
  `+ [INITIAL_INTERVAL=<interval> [<t0> <i0> [<t1> <i1> ...]]]`

To restart from an existing restart file, the file can be specified by using either the `FILE=<filename>` parameter to explicitly request a file or `JOB=<job name> START_TIME=<time>` to specify a file prefix and a specific time. The time must exactly match an output file time for the simulator to correctly load the file. To continue generating restart output files, `INITIAL_INTERVAL=<interval>` and following intervals can be appended to the command in the same format as described above.

- Example - Restart from checkpoint file at 0.133 μs:

```
.OPTIONS RESTART JOB=checkpt START_TIME=0.133us
```

- Example - Restart from checkpoint file at 0.133 μs :

```
.OPTIONS RESTART FILE=checkpt0.000000133
```

- Example - Restart from 0.133 μs and continue checkpointing at 0.1 μs intervals:

```
.OPTIONS RESTART FILE=checkpt0.000000133 JOB=checkpt_again
+ INITIAL_INTERVAL=0.1us
```

7.3 DC Analysis

The DC sweep analysis capability in **Xyce** carries out a sweep, in DC mode, on a circuit. DC sweep is supported for a source (current or voltage), through a range of specified

values. As the sweep proceeds, the bias point is computed for each value in the specified range of the sweep.

If the variable to be swept is a voltage or current source, a DC source must be used. The DC value is set in the netlist (see the **Xyce** Reference Guide [3]). In simulating the DC response of an analog circuit, **Xyce** eliminates any time dependence from the circuit. This is accomplished by treating all capacitor elements as open circuits, all inductor elements as short circuits and using only the DC values of both voltage and current sources.

.DC Statement

One specifies a `.DC` analysis with a `.DC` line in the netlist. Some examples of typical `.DC` lines are:

Example:

```
.DC M1:L 7u 5u -1u
.DC OCT V0 0.125 64 2
.DC DEC R1 100 10000 3
.DC TEMP LIST 10.0 15.0 18.0 27.0 33.0
```

The examples include all four types of sweep - linear, octave, decade, and list. For a complete description of each of these, see the **Xyce** Reference Guide [3].

Setting Up and Running a DC Sweep

Following the example given in Section 3.2, the diode clipper circuit netlist is shown in Figure 7.1 with a DC sweep analysis specified. Here, the voltage source `Vin` is swept from -10 to 15 in 1 volt increments, resulting in 26 DC operating point calculations. Note also that the default setting for `Vin` is ignored during these calculations. All other source values use the specified values (`VCC = 5V` in this case).

Running **Xyce** on this netlist produces an output results file named `clipper.cir.prn`. Obtaining this file requires that the `.PRINT DC` line be specified.Plotting this data produces the graph shown in Figure 7.2.

```
Diode Clipper Circuit
** Voltage Sources
VCC 1 0 5V
VIN 3 0 0V
* Analysis Command
.DC VIN -10 15 1
* Output
.PRINT DC V(3) V(2) V(4)
* Diodes
D1 2 1 D1N3940 D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
** GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE:  DIODE
* SUBTYPE:  RECTIFIER
.MODEL D1N3940 D(
+          IS = 4E-10
+          RS = .105
+           N = 1.48
+          TT = 8E-7
+         CJO = 1.95E-11
+          VJ = .4
+           M = .38
+          EG = 1.36
+         XTI = -8
+          KF = 0
+          AF = 1
+          FC = .9
+          BV = 600
+         IBV = 1E-4)
* .END
```

Figure 7.1. Diode clipper circuit netlist for DC sweep analysis.

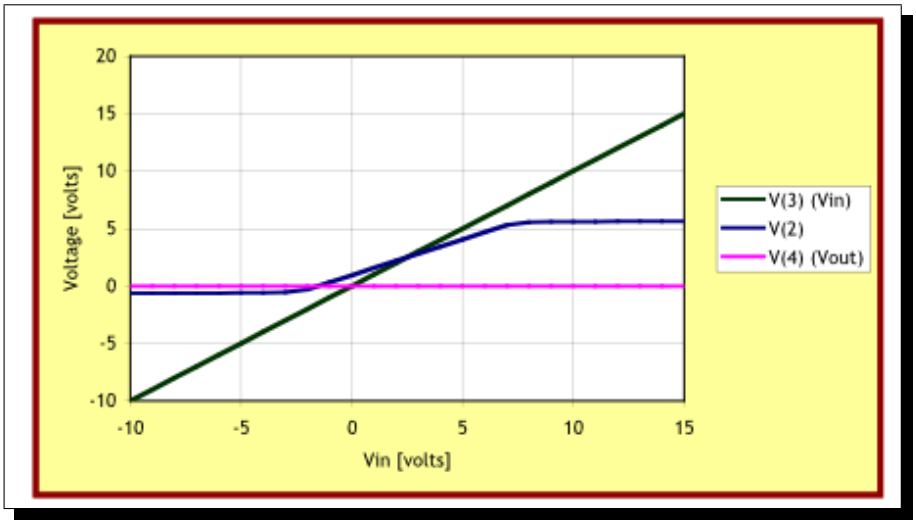

Figure 7.2. DC sweep voltages at Vin, node 2 and Vout.

OP Analysis

Xyce also supports `.OP` analysis statements. In **Xyce**, `.OP` should be considered as a shorthand for a single step DC sweep, in which all the default operating point values are used. One can also consider `.OP` analysis to be the operating point calculation which would occur as the initial step to a transient calculation, without the subsequent time steps.

This capability was mainly added so that the code would be able to handle legacy netlists which used this type of analysis statement. In most versions of SPICE, using `.OP` will result in extra output which is not available from a DC sweep. That additional output capability has not yet been implemented in **Xyce**.

7.4 STEP Parametric Analysis

The .STEP command performs a parametric sweep for all the analyses of the circuit. When this command is invoked, all of the typical analysis, such as .DC or .TRAN analysis are performed at each parameter step.

This capability is very similar to the STEP capability in PSPICE [4], but not identical. In **Xyce**, .STEP can be used to sweep over any device instance or device model parameter, as well as the circuit temperature. Currently, there is not a capability for sweeping global parameters, as specified by a .PARAM statement (Section 4.3).

.STEP Statement

One specifies a .STEP analysis by simply adding a .STEP line to a netlist. Note that unlike .DC, .STEP by itself is not an adequate analysis specification, as it merely specifies an outer loop around the normal analysis. There still needs to be a standard analysis line, either specifying .TRAN or .DC analysis.

Some examples of typical .STEP lines are:

Example:

```
.STEP M1:L 7u 5u -1u
.STEP OCT V0 0.125 64 2
.STEP DEC R1 100 10000 3
.STEP TEMP LIST 10.0 15.0 18.0 27.0 33.0
```

.STEP has a format similar to that of the .DC specification. In the first example, M1:L is the name of the parameter, 7u is the initial value of the parameter, 5u is the final value of the parameter, and -1u is the step size. Like .DC, .STEP in **Xyce** can also handle sweeps by decade, octave or specified lists of values. For a complete explanation of each type of sweep, consult the **Xyce** Reference Guide [3].

Sweeping over a Device Instance Parameter

The first example uses M1:L as the parameter, but it could have used any model or instance parameter that existed in the circuit. Internally, **Xyce** handles the parameters for all device models and device instances in the same way. You can uniquely identify any parameter

by specifying the device instance name, followed by a colon (:), followed by the specific parameter name. For example, all the MOSFET models have an instance parameter for the channel length, L. If you have a MOSFET instance specified in a netlist, named M1, then the full name for M1's channel length parameter is M1:L.

A simple application of .STEP to a device instance is given in figure 7.3. This is the same diode clipper circuit as was used in the transient analysis chapter, except that a single line (in red font) has been added. The .STEP line will cause **Xyce** to sweep the resistance of the resistor, R4, from 3.0 KOhms to 15.0 KOhms, in increments of 2.0 KOhms. This means that a total of seven transient simulations will be performed, each one with a different value for R4.

As the circuit is executed multiple times, the resulting output file is a little more sophisticated. The .PRINT statement is still used in much the same way as before. However, the .prn output file contains the concatenated output of each .STEP increment. For details of how .STEP changes output files, see the end of this section.

Sweeping over a Device Model Parameter

Sweeping a model parameter can be done in an identical manner to an instance parameter. Figure 7.4 contains the same circuit as in figure 7.3, but with a new .STEP line added. The new .STEP line refers to a model parameter, D1N3940:IS. Note that separate .STEP lines are the correct way to specify multiple parameters for a .STEP analysis. Each parameter needs its own separate line. In this respect, the .STEP line syntax differs from the .DC line syntax.

Sweeping over Temperature

It is also possible to sweep over temperature. To do so, simply specify temp as the parameter name. It will work in the same manner as .STEP when applied to model and instance parameters.

Special cases: Sweeping Independent Sources, Resistors, Capacitors

For some devices, there is generally only one parameter that one would want to actually sweep. For example, a linear resistor's only parameter of interest is the resistance, R. Similarly, for a DC voltage or current source, one is usually only interested in the magnitude of

```
Transient Diode Clipper Circuit with step analysis
* Voltage Sources
VCC 1 0 5V
VIN 3 0 SIN(0V 10V 1kHz)
* Analysis Command
.TRAN 2ns 2ms
* Output
.PRINT TRAN V(3) V(2) V(4)
 * Step statement
.STEP R4:R 3.0K 15.0K 2.0K
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE:  DIODE SUBTYPE:  RECTIFIER
.MODEL D1N3940 D(
+          IS = 4E-10
+          RS = .105
+           N = 1.48
+          TT = 8E-7
+         CJO = 1.95E-11
+          VJ = .4
+           M = .38
+          EG = 1.36
+         XTI = -8
+          KF = 0
+          AF = 1
+          FC = .9
+          BV = 600
+         IBV = 1E-4)
*
.END
```

Figure 7.3. Diode clipper circuit netlist for step transient analysis.

```
Transient Diode Clipper Circuit with step analysis
* Voltage Sources
VCC 1 0 5V
VIN 3 0 SIN(0V 10V 1kHz)
* Analysis Command
.TRAN 2ns 2ms
* Output
.PRINT TRAN V(3) V(2) V(4)
 * Step statements
.STEP R4:R 3.0K 15.0K 2.0K
.STEP D1N3940:IS 2.0e-10 6.0e-10 2.0e-10
* Diodes
D1 2 1 D1N3940
D2 0 2 D1N3940
* Resistors
R1 2 3 1K
R2 1 2 3.3K
R3 2 0 3.3K
R4 4 0 5.6K
* Capacitor
C1 2 4 0.47u
* GENERIC FUNCTIONAL EQUIVALENT = 1N3940
* TYPE:  DIODE SUBTYPE:  RECTIFIER
.MODEL D1N3940 D(
+          IS = 4E-10
+          RS = .105
+           N = 1.48
+          TT = 8E-7
+         CJO = 1.95E-11
+          VJ = .4
+           M = .38
+          EG = 1.36
+         XTI = -8
+          KF = 0
+          AF = 1
+          FC = .9
+          BV = 600
+         IBV = 1E-4)
*
.END
```


Figure 7.4. Diode clipper circuit netlist for 2-step transient analysis.

the source. Finally, linear capacitors generally only have the capacitance, C, as a parameter of interest. To make things easier for the user, these three types of devices have default parameters. Examples of usage are given below.

Example:

```
.STEP R4   3.0K   15.0K   2.0K
.STEP VCC  4.0     6.0    1.0
.STEP ICC  4.0     6.0    1.0
.STEP C1   0.45u   0.50u  0.1u
```

Independent sources require some extra explanation. There are a number of different types of independent sources, and only some of them have default parameters. Sources which are subject to `.DC` sweeps (swept sources) do not have a default parameter, as this could easily lead to infinite loops. The various independent source defaults are defined in table 7.2.

Source Type	Default
Sinusoidal source	V0 (DC value, Offset)
Exponential source	V1 (DC value, Initial value)
Pulsed source	V2 (Pulsed value)
Constant, or DC source	V0 (Constant value)
Piecewise Linear source	No default
SFFM source	No default
Swept source (specified on a `.DC` line)	No default

Table 7.2: Default parameters for independent sources.

Output files

A `.STEP` simulation can be thought of as several distinct executions of the same circuit netlist. The output data, as specified by a `.PRINT` line, however, goes to a single (`*.prn`) file. For convenience, a second auxilliary file is also created, with the `*.res` suffix.

The example file given in figure 7.3 has a filename of `clip.cir`. When run, it will produce files `clip.cir.res` and `clip.cir.prn`. `clip.cir.res` contains one line for each step, showing what parameter value was used on that step. `clip.cir.prn` is the familiar output format, but the `INDEX` field recycles to zero each time a new step begins. Since the default

behavior distinguishes each step's output only by recycling the INDEX field to zero, it can be beneficial to add the sweep parameters to the .PRINT line. For the default file format (format=std), these sweep parameters will not be included automatically, so for plotting purposes it is usually best to specify them by hand.

If using the default .prn file format (format=std), the output file resulting from a .STEP simulation will be a simple concatenation of each step's underlying analysis output. If using format=probe, the data for each execution of the circuit will be in distinct sections of the file, and it should be easy to plot the results using PROBE. If using format=tecplot, the results of each .STEP simulation will be in a distinct tecplot zone.

8. Using Homotopy Algorithms to Obtain Operating Points

Chapter Overview

This chapter includes the following sections:

8.1 Homotopy Algorithms Overview

The most difficult type of circuit problem to solve can be the DC operating point. Unlike transient solves, DC operating point analysis cannot rely on a good initial guess from a previous step, and also cannot simply reduce the step size when the solver fails. Additionally, operating points often have multiple solutions, sometimes intentional, and sometimes not. Multiple solutions can, even for circuit problems that converge, result in a standard Newton solve being unreliable. For example, it has been observed that the operating point solution to a Schmidt trigger circuit may change from one hardware platform to another.

Homotopy methods can often provide solutions to difficult nonlinear problems, even when conventional methods (i.e. Newton's method) fail, and they have been found often be useful when applied to circuit analysis [9] [10]. This chapter gives an introduction to the usage of homotopy algorithms (sometimes called continuation algorithms) in **Xyce**. For a more complete description of solver options, see the Xyce Reference Guide [3].

HOMOTOPY Algorithms Available in **Xyce**

There are several types of homotopy which are available in **Xyce**. Most are accessible by setting `.options nonlin continuation=1`, which allows the user to sweep existing device parameters (models and instances), as well as a few reserved artificial parameter cases. The most obvious natural parameter to use is the magnitude(s) of independent voltage or current sources, the choice of which is equivalent to "source stepping" in SPICE. A **Xyce** source-stepping example is given in section 8.3. Note that for some circuits (like the afore mentioned Schmidt trigger) source stepping will lead to turning points in the continuation.

"GMIN stepping", another well-known SPICE method, is also invoked with `.options nonlin continuation=1` (and other parameters), and is a special case in that the paramter being swept is artificial. An example of GMIN stepping is given in section 8.4.

A special **Xyce**-only homotopy is an algorithm which is designed specifically for MOSFET circuits [11]. This algorithm involves two internal MOSFET model parameters, one for the MOSFET gain, and the other for the nonlinearity of the current-voltage relationship. This algorithm is invoked with `.options nonlin continuation=2`. This algorithm has proved to be very effective in some large MOSFET circuits. A detailed example is given in section 8.2.

8.2 MOSFET Homotopy

Figure 8.1 contains a MOSFET homotopy example netlist. Note that this is a usage example - the circuit itself does not require homotopy to run. Circuits which are complex enough to require homotopy would not fit on a single page. The lines pertinent to the homotopy algorithm are highlighted in red.

Explanation of Parameters, Best Practice

Note that this example shows one set of options, but there are a number of other combinations of options that will work.

There are a number of "best practice" rules, which are illustrated by the example in figure 8.1. They are:

- `voltlim=0`. This is **not** required anymore; for previous versions of **Xyce** (version 3.0 and earlier) homotopy algorithms will usually break if used in conjunction with voltage limiting. As voltlim=1 is the default, it is not necessary to specify anything anymore.
- `continuation=2`. This specifies that we are using the special MOSFET homotopy. This is a 2-pass homotopy, in which first a parameter having to do with the gain is swept from 0 to 1, and then a parameter relating to the nonlinearity of the transfer curve is swept from 0 to 1.
- `initialvalue=0.0`. This is required.
- `maxvalue=1.0`. This is required.
- `stepcontrol=1` or `stepcontrol=adaptive`. This specifies that the homotopy steps are adaptive, rather than constant. This is recommended.
- `maxsteps=100`. This sets the maximum number of continuation steps for each parameter. For the special MOSFET continuation (which has 2 parameters), this means a maximum of 200 steps.
- `maxnliters=200`. This is the maximum number of nonlinear iterations, and has precedence over the similar number which can be set on the .options nonlin line.
- `aggressiveness=1.0`. This refers to the step size control algorithm, and the value of this parameter can be anything from 0.0 to 1.0. 1.0 is the most aggressive. In practice, try starting with this set to 1.0. If the solver fails, then reset to a smaller number.

```
THIS CIRCUIT IS A MOS LEVEL 1 MODEL CMOS INVERTER
.TRAN 20ns 30us 0 5ns
.PRINT tran v(vout) v(in) v(1)
.options timeint reltol=5e-3 abstol=1e-3


* HOMOTOPY Options

* voltim=0 is not needed anymore!
* .options device voltlim=0

.options nonlin continuation=2

.options loca stepper=0 predictor=0 stepcontrol=adaptive
+ initialvalue=0.0 minvalue=-1.0 maxvalue=1.0
+ initialstepsize=0.2 minstepsize=1.0e-4
+ maxstepsize=5.0 aggressiveness=1.0
+ maxsteps=100 maxnliters=200


VDDdev  VDD 0 5V
RIN IN 1 1K
VIN1  1 0  5V PULSE (5V 0V 1.5us 5ns 5ns 1.5us 3us)
R1    VOUT  0  10K
C2    VOUT  0  0.1p
MN1   VOUT  IN 0   0   CD4012_NMOS  L=5u W=175u
MP1   VOUT  IN VDD VDD CD4012_PMOS  L=5u W=270u
.MODEL cd4012_pmos PMOS
.MODEL cd4012_nmos NMOS
.END
```

Figure 8.1. Example MOSFET homotopy netlist.

8.3 Natural Parameter Homotopy

Figure 8.2 contains a natural parameter homotopy netlist. It is the same circuit as was used in figure 8.1, except that some of the parameters are different. As before, the lines pertinent to the homotopy algorithm are highlighted in red.

Explanation of Parameters, Best Practice

There are a few differences between the netlist in figure 8.1 and figure 8.2. They are:

- `continuation=1`. Sets the algorithm to use natural parameter homotopy.
- `conparam=VDDdev`. If using natural parameter homotopy, this is required. It sets which input parameter to perform continuation on. The parameter name is subject to the same rules as parameter used by the `.STEP` capability. (See section 7.4). In this case the parameter is the magnitude of the DC voltage source, VDDdev. For this type of voltage source, it was possible to use the default device parameter (see section 7.4)

Using the magnitudes of independent voltage and current sources is a fairly obvious approach. Unfortunately, it doesn't seem to work very well in practice.

8.4 GMIN Stepping

A type of homotopy commonly available in circuit simulators is GMIN stepping. A netlist example of GMIN stepping is shown in figure 8.3.

The name "GMIN stepping" can be somewhat confusing, as "GMIN" is also a user-specified device package parameter that one can set, which is unrelated to this algorithm. In this context, "GMIN" refers to resistors attached from circuit nodes to ground. The GMIN stepping algorithm involves attaching an artificial resistor to ground to every node in the circuit, with the intent of removing it later.

The continuation parameter is the conductance that is applied to the artificial resistors. Initially the conductance is very large, and is iteratively reduced until the artificial resistors have a very high resistance. At the end of the continuation, the resistors are removed from the problem. At this point, assuming the continuation has been successful, the original user-specified problem has been solved.

```
THIS CIRCUIT IS A MOS LEVEL 1 MODEL CMOS INVERTER
.TRAN 20ns 30us 0 5ns
.PRINT tran v(vout) v(in) v(1)
.options timeint reltol=5e-3 abstol=1e-3

* HOMOTOPY Options

* voltim=0 is not needed anymore!
* .options device voltlim=0

.options nonlin continuation=1

.options loca stepper=0 predictor=0 stepcontrol=1
+ conparam=VDDdev
+ initialvalue=0.0 minvalue=-1.0 maxvalue=5.0
+ initialstepsize=0.2 minstepsize=1.0e-4
+ maxstepsize=5.0 aggressiveness=1.0
+ maxsteps=100 maxnliters=200


VDDdev  VDD 0 5V
RIN IN 1 1K
VIN1  1 0  5V PULSE (5V 0V 1.5us 5ns 5ns 1.5us 3us)
R1    VOUT  0  10K
C2    VOUT  0  0.1p
MN1   VOUT  IN 0   0   CD4012_NMOS  L=5u W=175u
MP1   VOUT  IN VDD VDD CD4012_PMOS  L=5u W=270u
.MODEL cd4012_pmos PMOS
.MODEL cd4012_nmos NMOS
.END
```

Figure 8.2. Example natural parameter homotopy netlist.

```
THIS CIRCUIT IS A GMIN STEPPING EXAMPLE.
.TRAN 20ns 30us 0 5ns
.PRINT tran v(vout) v(in) v(1)
.options timeint reltol=5e-3 abstol=1e-3

* HOMOTOPY Options

* voltim=0 is not needed anymore!
* .options device voltlim=0

.options nonlin continuation=1

.options loca
+ stepper=natural
+ predictor=constant
+ stepcontrol=adaptive
+ conparam=GSTEPPING
+ initialvalue=4
+ minvalue=-4
+ maxvalue=4
+ initialstepsize=-2
+ minstepsize=1.0e-6
+ maxstepsize=1.0e+12
+ aggressiveness=0.01
+ maxsteps=200
+ maxnliters=20
+ voltagelist=DOFS


VDDdev  VDD 0 5V
RIN IN 1 1K
VIN1  1 0  5V PULSE (5V 0V 1.5us 5ns 5ns 1.5us 3us)
R1    VOUT  0  10K
C2    VOUT  0  0.1p
MN1   VOUT  IN 0   0   CD4012_NMOS  L=5u W=175u
MP1   VOUT  IN VDD VDD CD4012_PMOS  L=5u W=270u
.MODEL cd4012_pmos PMOS
.MODEL cd4012_nmos NMOS
.END
```

Figure 8.3. Example GMIN stepping netlist. Note that the continuation type is 1, and the continuation parameter is called GSTEPPING.

Explanation of Parameters, Best Practice

In general, GMIN stepping can be very useful, but in most cases it should not be the first choice of algorithm for the operating point. For large MOSFET circuits, the MOSFET-homotopy (continuation=2) has been observed to be much more effective. For non-MOSFET circuits, GMIN stepping may be a good candidate.

8.5 Pseudo Transient

Pseudo transient continuation is very similar to GMIN stepping, in that both algorithms involve placing large artificial terms on the Jacobian matrix diagonal, and progressively making these terms smaller until the original circuit problem is recovered. One difference is that, rather than doing a series on Newton solves, Pseudo transient does a single nonlinear solve while progressively modifying the pseudo transient parameter. An example of pseudo transient homotopy options is shown in figure 8.4.

Explanation of Parameters, Best Practice

Similar to GMIN stepping, pseudo transient can be useful, but has not been observed to be as successful as MOSFET-homotopy for large MOSFET circuits. Similar to GMIN Stepping, for difficult non-MOSFET circuits it may be a good candidate. It will not work as often as GMIN stepping, but when it does work, it tends to be much faster, as the total number of matrix solves is much smaller.

```
* HOMOTOPY Options
.options nonlin continuation=9

.options loca
+ stepper=natural
+ predictor=constant
+ stepcontrol=adaptive
+ initialvalue=0.0
+ minvalue=0.0
+ maxvalue=1.0e12
+ initialstepsize=1.0e-6
+ minstepsize=1.0e-6
+ maxstepsize=1.0e6
+ aggressiveness=0.1
+ maxsteps=200
+ maxnliters=200
+ voltagescalefactor=1.0
```

Figure 8.4. Example of Pseudo transient solver options. Note that the continuation parameter is set to 9.

This page is left intentionally blank

9. Time Integration and Multi-time Partial Differential Equations (MPDE)

Chapter Overview

This chapter provides an overview and guidance on the time integration algorithms in **Xyce**, Different time integration options can significantly effect simulation performance and accuracy. Additionally, this chapter provides an overview and using instructions for the new Multi-Time Partial Differential Equation (MPDE) capability in **Xyce**. This chapter includes the following sections:

- Section 9, *Differential Algebraic Equation (DAE) Time Integration*
- Section 9.2, *Multi-time Partial Differential Equations (MPDE) Overview*
- Section 9.3, *MPDE Usage*

9.1 Differential Algebraic Equation (DAE) Time Integration

Xyce version 4.0 includes a major upgrade to the time integration capability. This upgrade will result in transient simulations running more accurately and with substantially fewer computed time steps. This improved accuracy and efficiency comes at a small cost, as more data structures must be maintained in the code to support this capability. However, this cost has been minimized and the benefits of the new integrator should be substantial.

The old time integrator was similar to that of Spice3f5, and its descendents. Spice-style integrators are based around older ordinary-differential-equation (ODE) integrators, which assume every equation in the system of equations include time derivatives. In circuit problems, there are some fraction of the system of equations are purely algebraic. For example, the KCL equation for a voltage node that has only linear resistors attached to it is an algebraic equation, not a differential equation. Systems of equations that include some algebraic constraints are referred to as differential algebraic equations (DAE), and such systems (as can be found in circuit simulation) are better served by time integrators that have been specifically designed for DAE's.

Part of the reason for this is numerical stability. Systems of DAE's are often characterized by the DAE index, which is defined as "the minimum number of times that all or part of [the DAE] must be differentiated with respect to t" in order to reduce it to an ODE[8]. A system of index=0 is an ODE system. Many circuits have an index of 1 or more, and integrated MOSFET circuits usually have an index of at least 2 [12]. This is important because higher index systems are only stable and accurate for higher orders of integration, and are also very sensitive to initial conditions. In contrast, an ODE (index=0) integrator has the luxury of assuming that any implicit time integration scheme will be stable.

Solver Options and Guidance

In general, from a user point of view, the switch to the new time integrator should be transparent, and **Xyce** version 4.0 should behave the same with previous **Xyce** releases. Also, over the long term, the new time integrator should be a significant improvement over the old one. However, there will be some rare exception circuits in which the new time integrator will have more difficulty than the old one. This is mostly due to the fact that the new time integrator hasn't been in the code long enough to be completely mature. If a transient circuit that ran fine in previous releases does not run successfully in **Xyce** version 4.0, there are several options available.

- Loosen the time integrator tolerances. The default absolute and relative truncation error tolerances are the same as they were before, with the default abstol=1.0e-6 and the default reltol=1.0e-2. The new time integrator will generally get accurate results with looser tolerances, because the integration scheme is inherently more accurate and less subject to numerical dispersion. If a simulation exits with a time-step-too-small error, try using larger numbers for abstol and/or reltol. For example: `.options timeint abstol=1e-4 reltol=1e-2`.

- Change the maximum order. In the old time integrator, the maximum integration order by default was 1. In the new time integrator, the maximum order is, by default, 5. In some cases, this will make the simulation less robust. It is possible to set the maximum order=1, in the netlist by adding `.options timeint maxord=1`.

- If the first two options don't solve the problem, the last resort is to force **Xyce** to use the old time integrator. This is done by adding this to the netlist: `.options timeint newdae=0`. At the very least, if the circuit runs fine in a previous version of **Xyce**, this change will enable it to run in **Xyce** version 4.0.

9.2 Multi-time Partial Differential Equations (MPDE) Overview

Xyce version 4.0 includes a new analysis option specially designed for circuits with two disparate time scales. Normally, when a circuit has multiple time scales, the time integrator must take many small steps to fully resolve the fast time scale which can greatly slow simulation progress and reduce accuracy. Now, **Xyce** can discretize the system on the fast time scale converting the fast time DAE to a multi-time partial differential equation, hence MPDE. [13] This dramatically reduces the number of steps the time integrator must take resulting in a faster and more accurate simulation.

In general, best results will be obtained when there is a large difference between the fast and slow time scales. The quality of the solution of the fast time scale depends on the number of points used in the fast time discretization which also impacts simulation performance. Typically, if one needs n points to accurately discretize the fast time scale, then the fast and slow time scales should be separated by at least a factor of n, *i.e.* if one needed 10 points to represent the fast time solution, then best performance would be found when the fast and slow time scales differ by at lest a factor of 10. In an MPDE analysis, **Xyce** is effectively modeling n versions of the simulated circuit, so memory usage will grow proportionally with the number of points in the fast time domain.

9.3 MPDE Usage

To use MPDE analysis in a simulation, place the following option line in the netlist:

`.OPTIONS MPDE OSCSRC=<v1,v2...> IC=<1,2>` [other optional switches]

- `OSCSRC=...` A list of voltage and, or current sources that are to be considered as changing on the fast time scale. Typically these will be all the sources that change at or more quickly than the fast time scale.

- `IC=[1,2]` Specifies the method used to calculate the initial conditions for the MPDE problem. 1 uses the *Sawtooth* algorithm while 2 uses an initial transient run for the initial condition. During the initial condition calculation, the slow sources are deactivated so one is integrating only along the fast time axis.

- `N2=integer` Specifies the number of equally spaced points to use in discretizing the fast time scale. This option can only be used exclusive of the `AUTON2` and `AUTON2MAX` options.

- `AUTON2=[true | false]` False by default. If it is set to true, then an initial transient run is done to generate a *mesh* of time points for the fast time scale. The belief here is that the time integrator can do a better job picking out where it needs more points to accurately describe the fast time solution. Note: you can use `AUTON2` with `IC=1` or texttttIC=2. If you use it with `IC=1`, then the time mesh is just used for the set of points the for the Sawtooth initial conditions. This option can only be used exclusive to the `N2` option.

- `AUTON2MAX=integer` This sets the maximum number of fast time points that will be kept from the initial transient run so one has some control over the size of the simulation. If the initial transient run produces 2,000 points and this is set at 50, **Xyce** will uniformly sample points from the solution set for the MPDE simulation. This option can only be used exclusive to the `N2` option.

- `STARTUPPERIODS=integer` This is the number of fast time periods that **Xyce** should integrate through using normal transient analysis before trying to generate initial conditions for the MPDE analysis.

- `diff=[0,1]` Specifies the differentiation technique to use in calculating time derivatives on the fast time scale. 0 uses backward differences for the fast time differentiation while 1 uses central differences.

As an example here are two valid options lines for MPDE:

Example:
```
.options MPDE IC=1 N2=21 OSCSRC=Vsine
```

Example:
```
.options MPDE IC=2 AUTON2=true AUTON2MAX=50 OSCSRC=VIN1,VIN2
```

This page is left intentionally blank

10. Results Output and Evaluation Options

Chapter Overview

This chapter illustrates how to output simulation results to data or output files.

- Section 10.1, *Control of Results Output*
- Section 10.2, *Additional Output Options*
- Section 10.3, *Evaluating Solution Results*

10.1 Control of Results Output

Xyce supports only one solution output command, .PRINT. .PRINT is quite flexible, and supports several output formats.

.PRINT Command

The .PRINT command sends the analysis results to an output file. **Xyce** supports several options on the .PRINT line of netlists that control the format of the output. The syntax for the command is as follows:

- .PRINT <analysis type> [options] <output variable(s)>

Example:

```
.PRINT TRAN FILE=Output.prn V(3) I(R3) ID(M5) V(4)
.PRINT DC format=tecplot FILE=Output.dat V(2) {I(C3)+abs(V(4))*5.0}
```

Table 10.1 gives the various options currently available to the .PRINT command. Note that as of **Xyce** Release 3.0, it is possible to include device current or lead current specifications on the .PRINT line. Expressions can be output that include any valid .PRINT quantities. For further information, see the **Xyce** Reference Guide [3].

10.2 Additional Output Options

.OPTIONS OUTPUT Command

The main purpose of the .OPTIONS OUTPUT command is to provide control of the frequency at which data is written to files specified by .PRINT TRAN commands. This can be especially useful in controlling the size of the results file for simulations which required a large number of time steps. An additional benefit is that reducing the output frequency from the default, which outputs results at every time-step, can improve performance. The format for controlling the output frequency is:

- .OPTIONS OUTPUT INITIAL_INTERVAL=<interval> [<t0> <i0> [<t1> <i1> ...]]

Option...	Action...
FORMAT=<STD|NOINDEX|PROBE|RAW>	Controls the output format. The STD format outputs data in standard columns. The NOINDEX format is the same as the standard format except that the index column is omitted. The PROBE format specifies that the output should be formatted to be compatible with the PSpice Probe plotting utility. The RAW format specifies that the output conform to the Spice binary rawfile. Use the **-a** command line option to produce an ascii rawfile. The *default* is STD.
FILE=<output filename>	Allows the user to specify the output filename. The *default* is the netlist filename with the characters ".prn" appended (e.g., foo.cir.prn where foo.cir was the input netlist filename.
WIDTH=<print field width>	Allows the user to control the column width for the output data.
PRECISION=<floating point precision>	Controls the number of significant digits past the decimal point.
FILTER=<filter floor value>	Specifies the absolute value below which output variables will be printed as 0.0.
DELIMITER=<TAB|COMMA>	Specifies an alternate delimiter between columns of output in the STD output format.

Table 10.1. .PRINT command options.

where INITIAL_INTERVAL=<interval> specifies the starting interval time for output and <tx ix> specifies later simulation times (tx) where the output interval will change to (ix).

The following example shows the output being requested (via the netlist .OPTIONS OUTPUT command) every .1μs for the first 10μs, every 1μs for the next 10μs, and every 5μs for the remainder of the simulation:

Example:
```
.OPTIONS OUTPUT INITIAL_INTERVAL=.1us 10us 1us 20us 5us
```

Note: Using the old time integrator (e.g. running with the -newdae off command line option) **Xyce** will output data at the next time that is greater-than or equal to the current interval time. This means that output might not correspond exactly to the time intervals due to the adaptive time stepping algorithm. When using the new time integrator (e.g. when running with no -newdae command line option, or with -newdae on), **Xyce** will output at exactly the times you request in the output options; if the simulation advances past one or more requested output times in one integrator time step, **Xyce** will interpolate the solution to all requested time values that were passed.

10.3 Evaluating Solution Results

This section describes how to view graphical waveform analysis of the simulation results generated by **Xyce**. You can use the solution output features of **Xyce** in conjunction with graphing tools (e.g., TecPlot, gnuplot, MS Excel, etc.) to analyze graphically the waveform data created by a **Xyce** circuit simulation (see Figure 10.1 below for an example plot using TecPlot, http://www.amtec.com). In addition, by using the FORMAT=PROBE option to the .PRINT command, **Xyce** is able to output .csd files which can be read by the PSpice Probe utility to view the results. See the PSpice Users Guide [4] for instructions on using the Probe tool, and the **Xyce** Reference Guide [3] for details on the options to the .PRINT command.

Xyce produces two types of output: the simulation output file and the waveform data file. The calculations and results reported in the simulation output file can be thought of as an audit trail of the simulation. However, graphical analysis of data in the waveform data file is the most useful and accommodating way to evaluate simulation results.

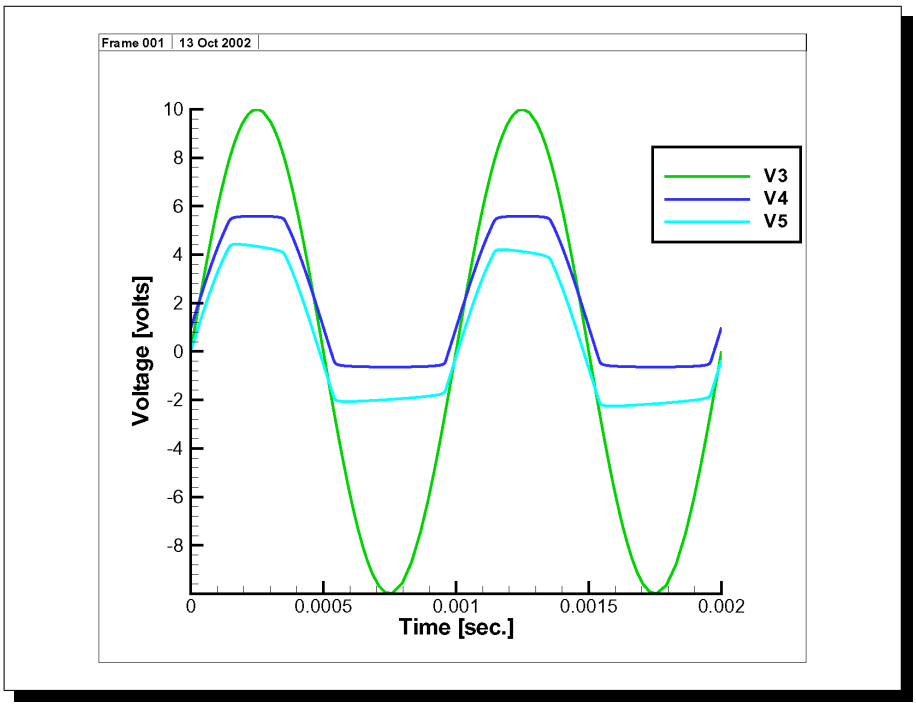

Figure 10.1. TecPlot plot of diode clipper circuit transient response from **Xyce** `.prn` file.

This page is left intentionally blank

11. Guidance for Running **Xyce** in Parallel

Chapter Overview

This chapter gives guidance on how to run a parallel version of **Xyce**. It includes the following sections:

- Section 11.1, *Introduction*
- Section 11.2, *Mechanics*
- Section 11.3, *Problem Size*
- Section 11.4, *Linear Solver Options*
- Section 11.5, *Partitioning Options*

11.1 Introduction

Xyce has been designed, from the ground up, to support a message-passing parallel implementation. As such, **Xyce** is unique among circuit simulation tools, and many of the issues pertinent to running in parallel are still research issues. However, **Xyce** is now mature enough that some general principles have emerged, for effectively running problems in a parallel environment.

11.2 Mechanics

Parallel simulations must be run from the command line. Details of how to do this are given in section 2.2.

11.3 Problem Size

Due to the overhead of interprocessor communication, running **Xyce** in parallel is only useful for large circuit problems. Also, for any problem size, there is an optimal number of processors. As one increases the processor count, the amount of communication required increases and the work per processor decreases. This increase in communication will slow a simulation down, while the reduction in work per processor will have the reverse effect. If the number of processors is too large, the benefit of distributing the problem will be outweighed by the high cost of communication overhead. Such a limit exists for every size of problem, and increasing the processor count beyond this point is counterproductive. This issue is most pronounced for platforms with a high communication cost, such as Beowulf clusters.

Smallest Possible Problem Size

Circuits are made up of a discrete set of components (voltage nodes, devices, etc.). To run in parallel, it is preferable that **Xyce** be able to put at least one discrete part of the problem on each processor. In practice, this means that the number of processors should be less than the number of nodes in the circuit. **Xyce** is capable of simulating smaller problems, but it is not recommended, due to solver instabilities.

Ideal Problem Size

To take full advantage of **Xyce**'s parallel capability, the problem should be relatively large. A good metric for estimating how many processors one should use is the number of devices per processor. The ideal number of devices per processor is machine and problem dependent. For machines such as SGI Challenger platforms, with relatively fast communication speeds in comparison to processor performance, reasonable speedups can be seen for 100's of devices per processor. For Beowulf clusters with relatively slow communication speeds in comparison to processor performance, 1000's of devices per processor are required to achieve reasonable speedups in parallel. These numbers are problem-dependent, as the effectiveness of the load balance and partitioning can vary substantially for different circuits.

11.4 Linear Solver Options

There are several different linear solvers available in **Xyce** version 4.1; they are:

- The AztecOO iterative solver library.
- KLU
- SuperLU.

For **Xyce** version 4.1, AztecOO is the only fully parallel solver available. However, both KLU and SuperLU are available with the parallel version of **Xyce**. With the later two solvers, the problem will be "assembled" in parallel, but the linear system will be solved in serial on processor 0. This can be quite effective for smaller parallel problems of a few thousand devices or less.

The solver can be specified through the `'.OPTIONS LINSOL'` control line in the netlist. By default, the parallel version of Xyce uses AztecOO as the linear solver. Conversely, a serial version of Xyce uses KLU as its default linear solver. To use a solver other than the default the user needs to add the option `'TYPE='<solver>` to the `'.OPTIONS LINSOL'` control line in the netlist, where `<solver>` is `'KLU'`, `'SUPERLU'`, or `'AZTECOO'`.

AztecOO

AztecOO is a parallel, iterative, linear solver package in Trilinos. The iterative solver used by Xyce is the Generalized Minimum Residual method, or GMRES. Some of the solver parameters for GMRES can be altered through the '.OPTIONS LINSOL' control line in the netlist. These parameters can have a significant impact on performance of **Xyce**, so consult the developer team if you have any questions. Table 11.1 has a list of solver parameters for AztecOO and their default values.

Option	Description	Default Value
AZ_max_iter	Maximum allowed iterations	500
AZ_tol	Iterative solver (relative residual) tolerance	1.0e-12
AZ_kspace	Krylov subspace size	500
AZ_ilut_fill	ILU fill level	2.0
AZ_drop	ILU drop tolerance	1.0e-3
AZ_overlap	ILU subdomain overlap	0
AZ_athresh	ILU absolute threshold	0.0001
AZ_rthresh	ILU relative threshold	1.0001
USE_IFPACK_PRECOND	Use ILU from Ifpack package	1 (true)
OUTPUT_LS	Write out linear systems to file every # solves	0

Table 11.1. AztecOO linear solver options.

If Xyce is run with the verbosity enabled for the linear algebra package, it is not uncommon to see warnings from AztecOO. These warnings usually indicate that the solver returned unconverged due to some numerical issue.

NOTE: Getting AztecOO warnings does *not* mean that the entire simulation has failed. **Xyce**, like all circuit simulators, uses a hierarchy of solvers, and if the iterative linear solver fails, that often means that the nonlinear solver or time integrator will then make adjustments and re-attempt the step. Getting warnings like the ones described here is usually not cause for concern, and the warnings can often be ignored. If the entire simulation eventually does fail (i.e. gets a ' 'time-step-too-small" error), then the warnings from AztecOO might contain clues as to what went wrong. However, by themselves they are usually benign.

Common AztecOO Warnings

The simplest reason for AztecOO to return unconverged would be when the maximum number of iterations was reached, which results in this warning:

```
************************************************************
Warning: maximum number of iterations exceeded without convergence
************************************************************
```

Another reason that AztecOO may return unconverged is when the GMRES Hessenberg matrix is ill-conditioned. This is usually a sign that the matrix and/or preconditioner is nearly singular. The resulting warning looks like:

```
************************************************************
Warning: the GMRES Hessenberg matrix is ill-conditioned.  This may
indicate that the application matrix is singular. In this case, GMRES
may have a least-squares solution.
************************************************************
```

It is also common to lose accuracy when the matrix and/or preconditioner are nearly singular. GMRES relies on an estimate of the residual norm, called the recursive residual, to determine convergence. The recursive residual is used instead of the actual residual for computational efficiency. However, numerical issues can cause the recursive residual to differ from the actual residual. When that situation is detected by AztecOO, and cannot be rectified, this warning will be outputted:

```
************************************************************
Warning: recursive residual indicates convergence
though the true residual is too large.

Sometimes this occurs when storage is overwritten (e.g. the
solution vector was not dimensioned large enough to hold
external variables). Other times, this is due to roundoff. In
this case, the solution has either converged to the accuracy
of the machine or intermediate roundoff errors occurred
preventing full convergence. In the latter case, try solving
again using the new solution as an initial guess.
************************************************************
```

KLU

KLU is a serial, sparse direct solver native to the Amesos package in Trilinos and is the default solver for serial versions of Xyce. KLU can be used in a parallel version of Xyce as well, but this requires the linear system to be solved on one processor and the solution communicated back to all processors. As long as the linear system can fit on one processor, KLU is often a superior approach to solving linear systems in parallel.

Some of the solver parameters for KLU can be altered through the '.OPTIONS LINSOL' control line in the netlist. These parameters can have a significant impact on performance of **Xyce**, so consult the developer team if you have any questions. Table 11.2 has a list of solver parameters for KLU and their default values.

Option	Description	Default Value
KLU_REPIVOT	Recompute pivot order each solve	1 (true)
OUTPUT_LS	Write out linear systems to file every # solves	0
OUTPUT_FAILED_LS	Write out failed linear systems to file	0 (false)

Table 11.2. KLU linear solver options.

SuperLU

SuperLU is a serial, sparse direct solver that has an interface in the Amesos package. Similar to KLU, SuperLU can be used in a parallel version of Xyce, but the linear system is solved on one processor. Xyce does not allow any modification to SuperLU's solver parameters at this time.

11.5 Partitioning Options

Xyce currently has graph partitioning available. This partitioning subdivides the circuit problem into sections that are then distributed to the processors. A good partition can have a dramatic effect on the parallel performance of circuit simulation. There are two key components to a good partition:

- Effective load balance.

- Minimizing communication overhead.

An effective load balance ensures that the computational load of the calculation is equally distributed among the available processors. Minimizing communication overhead seeks to distribute the problem in a way that reduces the impact of underlying message passing during the simulation run. For runs with a small number of devices per processor the communication overhead becomes the critical issue, while for runs with larger numbers of devices per processor the load balancing becomes more important. **Xyce** has integrated within it the **ZOLTAN** library of parallel partitioning heuristics.

Zoltan Partitioning of the Linear System

Zoltan can be controlled through the `'.OPTIONS LINSOL'` control line in the netlist. For parallel builds of **Xyce**, Zoltan is available and enabled by default. With Zoltan enabled, the linear system is statically load balanced, at the beginning of the simulation, based on the graph of the Jacobian matrix. The local system is also reordered based on nested dissection which should improve conditioning and minimize fill.

These techniques can be very effective for improving the efficiency of the iterative linear solvers. See the **Zoltan User Guide** [14] for more details.

Occasionally it can be desirable to turn off the partitioning option, even for parallel simulations. To do so, the users should add the option `'.OPTIONS LINSOL TR_PARTITION=0'` to disable Zoltan. In general, this is only recommended if the user wants to run a very small circuit in parallel.

Singleton Filtering of the Linear System

Singleton filtering refers to the reduction of the linear system through removal of all rows and columns with single non-zero entries. The values associated with these removed entries can be resolved as pre/post solve linear operations. A by-product of this reduction is a more tractable, sparser linear system for both the load balancing and linear solver algorithms. This functionality is turned on by adding the `'TR_SINGLETON_FILTER=1'` option to the `'.OPTIONS LINSOL'` control line in the netlist.

An important caveat is that the performance and convergence of the linear solver for parallel problems can be substantially less robust for some circuits. This is a known issue with parallel iterative solution of linear problems. If **Xyce** is not performing as expected for these problems, please consult the developer team.

This page is left intentionally blank

12. Handling Power Node Parasitics

Chapter Overview

This chapter includes the following sections:

- Section 12.1, *Power Node Parasitics*
- Section 12.2, *Two Level Algorithms Overview*
- Section 12.3, *Examples*
- Section 12.4, *Restart*

12.1 Power Node Parasitics

Parasitic elements (R, L, C) are frequently required for circuit simulations to capture important circuit behavior. Most parasitic elements (interconnect, etc.) can be added to netlists without causing any difficulties for the **Xyce** solvers. Small circuits in particular are very robust to the addition of parasitic elements. Larger circuits, however, that must be simulated in parallel will in general tend to have more solver difficulties with the addition of parasitic devices. Of particular note are parasitic elements attached to the power and/or ground nodes of large digital circuits. As these nodes tend to be highly connected, they can potentially have very high impact on solver difficulties.

One of the parallel algorithms used by **Xyce** is called *singleton removal*. This algorithm is applied at the linear solver level and is crucial for getting many large circuits to run in parallel. This algorithm takes advantage of the fact that, in circuit simulation, some solution values are available *explicitly*, rather than being a quantity that needs to be calculated as the solution to a particular equation. In circuit simulation, such quantities are usually the values of independent sources. For instance, the presence of an independent voltage source at a particular node in a circuit fixes the voltage at that node to be the value of the independent source; therefore, equations reflecting the value of the voltage at that particular node do not have to be added to the set of linear equations that are used (in part) to determine the voltages at all nodes in the circuit. The technique of fixing such node voltages without including them in the rest of the linear solve can be handled in a preprocessing phase referred to as the singleton removal phase.

When simulating in parallel, singleton removal is crucial since some voltage sources (especially power supplies in digital circuits) are connected to hundreds or thousands of circuit nodes. In parallel, this presents a big problem because a large number of connections can often mean a communication bottleneck during the linear solve. The use of singleton removal eliminates that bottleneck.

While singleton removal can result in a great improvement for circuits with ideal power supplies, for circuits with non-ideal power supplies, the communication bottleneck remains. Once parasitic elements are placed between the power supply and the rest of the circuit, it is only the voltage at the circuit node which is directly connected to the independent source that can be removed via singleton removal. All of the other nodes that are connected to this independent source through parasitic elements have voltages that must now be solved for directly.

12.2 Two Level Algorithms Overview

Fortunately there is a workaround in **Xyce** which allows power node parasitics to be included in large circuits without breaking singleton removal. The workaround requires the use of a two-level Newton solve, in which the problem is broken up into two very separate pieces. Each piece is, for the most part, treated as an entirely separate circuit with minimal coupling terms linking the pieces together.

For power-node problems, two-level users will typically split the netlist into "top" and "inner" netlists. The top netlist should contain the power node parasitics and the ideal voltage sources, and very little else. The inner circuit should contain the rest of the circuit. The two circuits are coupled through a "EXT" (external) device in the top circuit, and two or more independent voltage sources on the inner circuit. The values on the inner voltages are imposed from the top circuit, and the currents and conductances of the EXT device come from the inner circuit.

Xyce will treat the two circuits separately, constructing a different linear system for each one. As such, the inner circuit will appear to have independent sources, and the singleton removal algorithm will still work.

The two-level Newton algorithm has been in the literature since (at least) the 1980's, although in the past it has mostly been applied to circuit-device simulation. For a mathematical description, see [15] and [16].

12.3 Examples

Explanation and Guidance

An example of a circuit that uses the two level algorithm is given in figures 12.1 and 12.2. The top circuit (compTop.cir) is given in the first figure, and this top circuit invokes the inner circuit (compInner.cir) with the extern device, `y1`. To run this circuit, the user will only specify the top circuit on the command line:

```
Xyce compTop.cir <return>
```

The extern device (`YEXT y1` sits between the contents of compTop.cir and compInner.cir. It is connected to two nodes in the top level circuit, `DD1` and `SS1`. From the perspective of

```
THIS CIRCUIT IS THE TOP PART OF A TWO LEVEL EXAMPLE.
* compTop.cir - BSIM3 Transient Analysis

YEXT y1 DD1 SS1 externcode=xyce netlist=compInner.cir
Vdd DDorig 0 5.0
Vss SSorig 0 0.0

.options linsol type=klu
.options timeint abstol=1.0e-6  reltol=1.0e-3

* PARASITICS
l_Lwirevdd    DDorig Ny   .50n
l_Lwirevss    SSorig Nx   .50n
R_Rbw         Ny     DD1    50m
R_Rwi         Nx     SS1    50m

.tran 0.01ns 60ns
.print tran v(DD1) v(SS1) i(Vdd)

.END
```

Figure 12.1. Example two-level top netlist.

```
THIS CIRCUIT IS THE INNER PART OF A TWO LEVEL EXAMPLE.
* compInner.cir - BSIM3 Transient Analysis

M1 Anot    A       DD1 DD1  PMOS w=3.6u l=1.2u
M2 Anot    A       SS1 SS1  NMOS w=1.8u l=1.2u
M3 Bnot    B       DD1 DD1  PMOS w=3.6u l=1.2u
M4 Bnot    B       SS1 SS1  NMOS w=1.8u l=1.2u
M5 AorBnot SS1     DD1 DD1  PMOS w=1.8u l=3.6u
M6 AorBnot B       1   SS1  NMOS w=1.8u l=1.2u
M7 1       Anot    SS1 SS1  NMOS w=1.8u l=1.2u
M8 Lnot    SS1     DD1 DD1  PMOS w=1.8u l=3.6u
M9 Lnot    Bnot    2   SS1  NMOS w=1.8u l=1.2u
M10 2      A       SS1 SS1  NMOS w=1.8u l=1.2u
M11 Qnot   SS1     DD1 DD1  PMOS w=3.6u l=3.6u
M12 Qnot   AorBnot 3   SS1  NMOS w=1.8u l=1.2u
M13 3      Lnot    SS1 SS1  NMOS w=1.8u l=1.2u
MQLO 8     Qnot    DD1 DD1  PMOS w=3.6u l=1.2u
MQL1 8     Qnot    SS1 SS1  NMOS w=1.8u l=1.2u
MLTO 9     Lnot    DD1 DD1  PMOS w=3.6u l=1.2u
MLT1 9     Lnot    SS1 SS1  NMOS w=1.8u l=1.2u

CQ Qnot 0 30f
CL Lnot 0 10f

Vconnect0000 DD1 0 0
Vconnect0001 SS1 0 0

Va A 0  pulse(0 5 10ns .1ns .1ns 15ns 30ns)
Vb B 0 0

.model nmos nmos (level=9)
.model pmos pmos (level=9)

.options linsol  type=klu
.options timeint abstol=1.0e-6 reltol=1.0e-3

.tran 0.01ns 60ns
.print tran v(a) v(b) 1.0+v(9) 1.0+v(8)

.END
```


Figure 12.2. Example two-level inner netlist.

compTop.cir, the YEXT y1 device just looks like a nonlinear two-terminal resistor, which is the equivalent of the entire inner circuit.

In the inner circuit, the nodes DD1 and SS1 are applied though the independent sources Vconnect0000 and Vconnect0001. By convention, the inner circuit must contain and independent voltage source for each node to which the EXT device is connected. The default naming convention requires that these sources be named vconnectxxxx, with xxxx being a four digit integer starting at 0000.

Note that the .tran statement on the inner circuit must match the .tran statement on the top circuit. The same is true for .DC analysis.

Note also that both circuit files have their own .print statements. That means that they will both produce *.prn output files.

The coupling between the top and inner layers requires extra linear solves, so when using this algorithm the code will run more slowly. In general, one can usually expect a factor of two slowdown, for circuits that can be run either as conventional or two-level simulations. So, in practice this algorithm should only be applied when it is really needed (i.e., when conventional simulations fail).

Finally, when using this method, one must take particular care with file names. In practice, a **Xyce** user may frequently change netlist file names to reflect new details about the run. When this happens, the name of the netlist invoked on the YEXT y1 line must be changed. Failure to do so may result in using the wrong file for the inner simulation.

12.4 Restart

Restart works with the two-level algorithm. However, as the two-level algorithm involves two separate netlist input files, two-level restart requires a separate restart file for each phase of the problem. So, the two files (for example compTop.cir and compInner.cir) need to have .options restart statements, and the statements in the two files need to be consistent with each other.

Currently, the code does *not* make any attempt to check if the ".options restart" statement in the top file is consistent with the ".options restart" statement in the inner file. It is up to the user to enforce this.

13. Specifying Initial Conditions

Chapter Overview

This chapter includes the following sections:

- Section 13.1, *Initial Conditions Overview*
- Section 13.2, *Device Level IC= Specification*
- Section 13.3, *.IC and .DCVOLT Initial Condition Statements*
- Section 13.4, *.NODESET Initial Condition Statements*
- Section 13.5, *.SAVE Statements*
- Section 13.6, *DCOP Restart*
- Section 13.7, *UIC and NOOP*

13.1 Initial Conditions Overview

There are several different initial condition options available in **Xyce**. There are several reasons why a user may want to set an initial condition. These include, but are not limited to:

- Improving the robustness of the DCOP solution.
- Optimizing performance by reusing DCOP solution of a previous run to start new transient runs.
- Setting an initial state for a digital circuit.
- Initiating an oscillator circuit.

As noted, setting initial conditions can be particularly useful for multi-state digital circuits, to preset the initial state. An example result which demonstrates how initial conditions can be used to set the state of a digital circuit is shown in Fig. 13.1. In this case, obtaining the state purely through transient simulation can take a very long time and often is not practical.

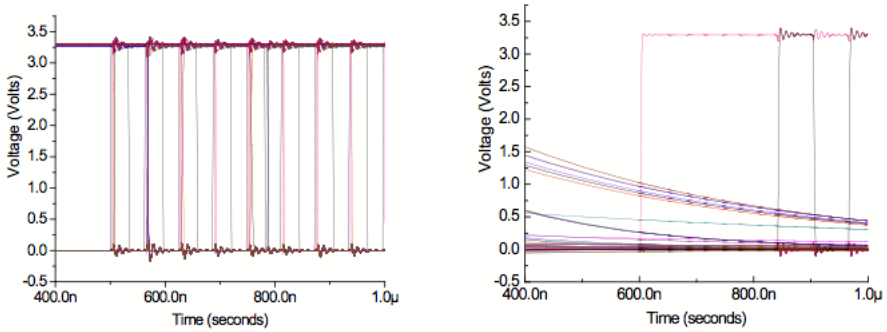

Figure 13.1. Example result with (left) and without (right) IC= preset. The preset example starts in the initial state directly out of the DCOP calculation, while the non-preset example requires a long transient to equilibrate.

13.2 Device Level IC= Specification

Many devices in **Xyce** support setting initial junction voltage conditions on the device instance line with the `IC=` keyword. This has frequently been used to set the state of digital circuits. A very simple inverter example that demonstrates usage of `IC=` on a BSIMSOI device is shown in Fig. 13.2.

While many circuit simulators have a similar `IC=` capability, the **Xyce** implementation differs in some important respects. For any device that has an `IC=` statement, **Xyce** enforces the junction drop in parallel with the device junction as a voltage source in parallel with the device. This parallel voltage source is applied through the DCOP calculation, and is then removed prior to the beginning of the transient. This strongly enforces the requested junction drop, meaning that if the DCOP converges, the requested voltage drop will in the solution. In many other circuit codes, `IC=` is applied as a weaker constraint, with the intent of improving DCOP calculation robustness.

Currently, `IC=` can be applied to the following devices: BSIM3, BSIM4, BSIMSOI, Capacitor and Inductor. This capability may be added to other devices in the future, depending on user requests.

```
MOS LEVEL=10 INVERTER WITH IC=

.subckt INV IN OUT VDD GND
MN1 OUT IN GND GND GND NMOS w=4u  l=0.15u IC=2,0
MP1 OUT IN VDD GND VDD PMOS w=10u l=0.15u
.ends

.tran 20ns 30us
.print tran v(vout) v(in)+1.0 v(1)

VDDdev  VDD 0 2V
RIN IN  1 1K
VIN1  1 0  2V PULSE (2V 0V 1.5us 5ns 5ns 1.5us 3.01us)
R1    VOUT  0  10K
C2    VOUT  0  0.1p
XINV1 IN VOUT VDD 0 INV
.MODEL NMOS NMOS ( LEVEL = 10 )
.MODEL PMOS PMOS ( LEVEL = 10 )

.END
```

Figure 13.2. Example netlist with device-level IC=.

13.3 .IC and .DCVOLT Initial Condition Statements

.IC and .DCVOLT are equivalent methods for specifying initial conditions. How they are applied depends on whether the UIC parameter is present on the .TRAN line. If UIC is not specified, then the conditions specified by a .IC and .DCVOLT statements are enforced throughout the DCOP phase, insuring that the specified values will be the solved values at the end of the DCOP calculation. Any unspecified variables are allowed to find their computed values, consistent with the imposed voltages.

```
RC circuit
.ic v(1)=1.0
c1 1 0 1uF
R1 1 2 1K
v1 2 0 0V
.print tran v(1)
.tran 0 5ms
.options timeint reltol=1e-6 abstol=1e-6
.end
```

Figure 13.3. Example netlist with .IC. Without the .IC statement, the capacitor is not given an initial charge, and the signals in transient are all flat. With the .IC statement, it has an initial change which then decays in transient.

If UIC is specified on the .TRAN line, then the DCOP calculation is skipped altogether, and the values specified on .IC and .DCVOLT lines are simply used as the initial values for the transient calculation. Any unspecified values are set to zero.

For both the UIC and non-UIC cases, any specified values that do not correspond to existing circuit variables are ignored. Also, currently the .IC and .NODESET capability can only set voltage values, not current values.

Syntax

```
.IC V(node1) = val1 <V(node2) = val2> ...
.DCVOLT V(node1) = val1 <V(node2) = val2> ...
```

where: *val1, val2, ...* specify nodal voltages and *node1, node2, ...* specify node numbers.

Example

```
.IC V(1) = 2.0  V(A) = 4.5
.DCVOLT  1 2.0 A 4.5
```

A more complete example (showing a full netlist) is given in Fig. 13.3.

13.4 .NODESET Initial Condition Statements

.NODESET is similar to .IC, except that **Xyce** enforces the specified conditions less strongly. For .NODESET simulations, **Xyce** does *two* nonlinear solves for the DCOP condition. For the first solve, the .NODESET values are enforced throughout the solve, similar to .IC. For the second solve, the result of the first solve is used as an initial guess, and all the values are allowed to float and eventually obtain their unconstrained, self-consistent values. As such, the computed values will not necessarily match the specified values. .NODESET is generally used to help with difficult nonlinear solves, but should only be used if the user wants a fully self-consistent initial DCOP computation. For an inconsistent initial condition, one should use .IC instead.

```
.NODESET V(node1) = val1 <V(node2) = val2> ...
.NODESET node1 val1 <node2 val2>
```

where: *val1, val2, ...* specify nodal voltages and *node1, node2, ...* specify node numbers.

Example

```
.NODESET V(1) = 2.0  V(A) = 4.5
.NODESET  1 2.0 A 4.5
```

13.5 .SAVE Statements

Operating point information can be stored using the .SAVE statements, and then reused to start subsequent transient simulations. Using .SAVE will result in solution data being stored in a text file, comprised of .NODESET or .IC statements. This file can be applied to other simulations using .INCLUDE.

The form of .SAVE is as follows:

```
.SAVE <TYPE=type_keyword> <FILE=save_file> <LEVEL=level_keyword> <TIME=save_time>
```

where:

type_keyword can be set to "NODESET" or "IC". By default, it will be "IC".

save_file is the user specified output file name for the *.ic file. If this is not specified, **Xyce** will use *netlist.cir*.ic.

level_keyword is an Hspice compatibility parameter. Currently, **Xyce** only supports "ALL" and "NONE".

save_time is an Hspice compatibility parameter. Currently it is unsupported, and **Xyce** can only output the *.ic file at time=0.0.

13.6 DCOP Restart

DCOP restart is a capability that is very similar to .NODESET and .SAVE used in combination. Similar to .NODESET, starting a simulation with DCOP restart will result in **Xyce** performing two nonlinear solves. The first solve strictly enforces the previous answer, and the second solve allows all the values to float and obtain their unconstrained solution. The second solve relies on the results of the first solve as an initial guess.

If the UIC keyword appears on the .TRAN line, then the contents of the DCOP restart file will be applied as the initial condition and the DCOP calculation will be skipped altogether. This is, of course, the same as for .IC and .NODESET.

While .NODESET and DCOP restart are very similar, there are a few differences. The biggest difference is the handling of voltage source voltages and currents. DCOP restart will attempt to restart from all the variables of the simulation, including voltage source variables. .NODESET, on the other hand, will ignore specified variables associated with voltage sources, and explicitly does not allow currents to be set. In general, voltage sources are constraints in and of themselves, so re-constraining them can cause singular matrices. To avoid this issue, DCOP restart, makes changes to the linear system to prevent matrices from being singular.

Saving a DCOP restart file

To create a DCOP restart file, a .DCOP output=filename line needs to be added to the netlist:

```
.dcop output=saved.op
```

This will result in the file "saved.op" being produced immediately after the operating point calculation. The produced file is similar to a *.ic file that can be produced by .SAVE, but with different format. Similar to the the *.ic file, it is a text file, and has two columns. One column has the variable name, the other column the value of that variable.

Loading a DCOP restart file

To use a DCOP restart file, a .DCOP input=filename line needs to be added to the netlist:

```
.dcop input=saved.op
```

If the specified file does not exist in the local directory, then **Xyce** will simply ignore the `.DCOP` statement and run normally.

13.7 UIC and NOOP

As noted earlier, the UIC key word on the TRAN line will disable the DCOP calculation, and will result in **Xyce** immediately going to transient. If .IC, .NODESET, or .DCOP input is specified, then the transient calculation will use the specified initial values as the initial starting point. The NOOP keyword works in exactly the same way as UIC.

```
pierce oscillator
c1 1 0 100e-12
c2 3 0 100e-12
c3 2 3 99.5e-15
c4 1 3 25e-12
l1 2 4 2.55e-3
r1 1 3 1e5
r2 3 5 2.2e3
r3 1 4 6.4
v1 5 0 12
Q1 3 1 0 NBJT
.MODEL NBJT NPN (BF=100)
.print tran  v(2) v(3)

.tran 1ns 1us  UIC
.ic v(2)=-10000.0 v(5)=12.0
```

Figure 13.4. Example netlist with UIC. This circuit is a pierce oscillator, and it will only oscillate if the operating point is skipped. This oscillator will take a really long time to achieve its steady-state amplitude if the .IC statement is not included. By including the .IC statement, the amplitude of node 2 is preset to a value close to its final steady-state amplitude. Note, the transient in this example only goes for 10 cycles as a demonstration. In general, the time scales for this oscillator are much longer than that and require millions of cycles.

Example

```
.tran 1ns 1us  UIC
.tran 1ns 1us  NOOP
```

Some circuits, particularly oscillator circuits, will only function properly if the operating point is skipped, as they need an inconsistent initial state to oscillate. A pierce oscillator example is given in Fig. 13.4.

14. Working with .PREPROCESS Commands

Chapter Overview

This chapter includes the following sections:

- Section 14.1, *Introduction*
- Section 14.2, *Ground Synonym Replacement*
- Section 14.3, *Removal of Unused Components*
- Section 14.4, *Adding Resistors to Dangling Nodes*

14.1 Introduction

In an effort to make **Xyce** more compatible with other commercial circuit simulators (e.g., HSPICE), some optional tools have been added to increase **Xyce**'s netlist processing capabilities. These options, which occur toward the beginning of a simulation, have been incorporated not only to make **Xyce** more compatible with different (i.e. non-**Xyce**) netlist syntax, but also to help detect and remove certain singular netlist configurations that can often cause a **Xyce** simulation to fail. Because all of the commands we describe below occur as a precursory step to setting up a **Xyce** simulation, they are all invoked in a netlist file via the keyword `.PREPROCESS`. In this chapter, we describe each of the different functionalities that can be invoked via a `.PREPROCESS` statement in detail and provide examples to illustrate its use.

14.2 Ground Synonym Replacement

In certain versions of Spice, keywords such as `GROUND`, `GND`, and `GND!` can be used as node names in a netlist file to represent the ground node of a circuit. **Xyce**, however, only recognizes node `0` as an official name for ground. Hence, if any of the prior node names is encountered in a netlist file, **Xyce** will treat these as different nodes from ground. To illustrate this point, consider the netlist of Fig. 14.1. When the node `Gnd` is encountered in the definition of resistor `R3`, **Xyce** instantitates this as a new node. The schematic diagram corresponding to this netlist (depicted in Fig. 14.2) shows that the resistor `R3` is "floating" between node `2` and a node which has only a single device connection, node `Gnd`. When the netlist of Fig. 14.1 is executed in **Xyce**, the voltage `V(2)` will evaluate to 0.5V.

If it is the case that one wishes for `Gnd` to be treated the same as node `0` in the above example, one can use the netlist of Fig. 14.3 instead. The statement `.PREPROCESS REPLACEGROUND TRUE` has the following effect: if this statement is present in a netlist, **Xyce** will treat any nodes named `GND`, `GND!`, `GROUND`, or any capital/lowercase variant of these keywords (e.g., `gROunD`) as synonyms for node `0`. Hence, according to **Xyce**, the netlist of Fig. 14.3 corresponds to the schematic diagram of Fig. 14.4, and the voltage `V(2)` will evaluate to 0.33V.

A few comments are in order. First, only one `.PREPROCESS REPLACEGROUND` statement is allowed per netlist file (this is to prevent the user from setting `REPLACEGROUND` to `TRUE` on one line and then to `FALSE` on another). Secondly, there is currently no way to differentiate between different keywords, i.e., it is not possible to treat `GROUND` as a synonym for node `0` while allowing `GND` to represent an independent node. If `REPLACEGROUND` is set to `TRUE`, *both* of these keywords will be treated as node `0` if present in a netlist file.

```
Circuit with "floating" resistor R3

V1 1 0 1
R1 1 2 1
R2 2 0 1
R3 2 Gnd 1

.DC V1 1 1 0.1
.PRINT DC V(2)
.END
```

Figure 14.1. Example netlist where `Gnd` is treated as being *different* from node `0`.

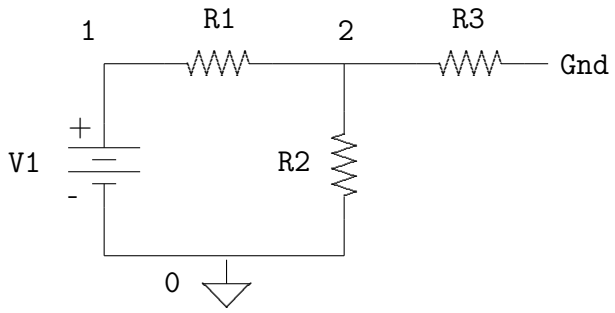

Figure 14.2. Circuit diagram corresponding to the netlist of Fig. 14.1 where node `Gnd` is treated as being *different* from node `0`.

```
Circuit where resistor R3 does *not* float

V1 1 0 1
R1 1 2 1
R2 2 0 1
R3 2 Gnd 1

.PREPROCESS REPLACEGROUND TRUE

.DC V1 1 1 0.1
.PRINT DC V(2)
.END
```

Figure 14.3. Example netlist where `Gnd` is treated as a synonym for node `0`.

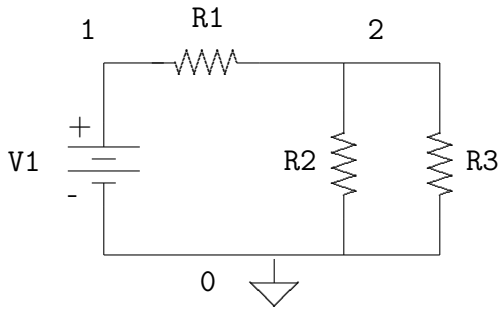

Figure 14.4. Circuit diagram corresponding to Fig. 14.3 where node `Gnd` is treated as a synonym for node `0`.

14.3 Removal of Unused Components

Consider a slight variant of the circuit in Fig. 14.3 with the netlist given in Fig. 14.5. Here, the resistor R3 is connected in a peculiar configuration: both terminals of the resistor are tied to the same circuit node, as is illustrated in Fig. 14.6. Clearly, the presence of this resistor has no effect on the other voltages and currents in the circuit since, by the very nature of its configuration, it has no voltage across it and, hence, does now draw any current. Therefore, in some sense, the component can be considered as "unused". It should be noted here that the presence of a resistor such as R3 is rarely/never introduced by design. Most times, the presence of such components is the result of either human or automated error.

```
Circuit with an unused resistor R3

V1 1 0 1
R1 1 2 1
R2 2 0 1
R3 2 2 1

.DC V1 1 1 0.1
.PRINT DC V(2)
.END
```

Figure 14.5. Netlist with a resistor R3 whose device terminals are both the same node (node 2).

While the presence of the resistor R3 in the above example does not change the behavior of the circuit, it does add an additional component to the netlist which **Xyce** must deal with when solving for the voltages and currents in the circuit. If the number of such components in a given netlist is large, it is potentially desirable to remove them from the netlist to ease the burden on **Xyce**'s solver engines. This, in turn, can help to avoid possible convergence issues. For example, even though the netlist in Fig. 14.5 will run properly in **Xyce**, the netlist of Fig. 14.7 will abort. The voltage source V2 attempts to place a 1V difference between its two device terminals; however, since both nodes of the voltage source are the same, the voltage source is effectively shorted.

In order to prevent situations such as the above from occuring, **Xyce** includes the command:

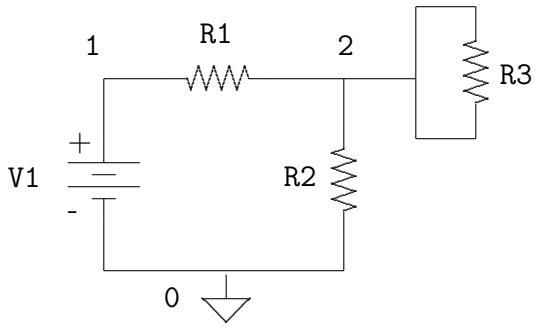

Figure 14.6. Circuit of Fig. 14.5 containing a resistor R3 whose terminals are tied to the same node (node 2).

```
Circuit with improperly connected voltage source V2

V1 1 0 1
R1 1 2 1
R2 2 0 1
V2 2 2 1

.DC V1 1 1 0.1
.PRINT DC V(2)
.END
```

Figure 14.7. Circuit with an improperly connected voltage source V2.

```
.PREPROCESS REMOVEUNUSED <component list>
```

where `<component list>` is a list of device types separated by commas. For each device type specified in the list, **Xyce** will check for instances of that device type for which all of the device terminals are connected to the same node. If such a device is found, **Xyce** will remove that device from the netlist. For instance, if we execute the netlist of Fig. 14.8, **Xyce** will seek out resistors and capacitors whose device terminals are connected to the same node and remove them from the netlist. This causes the resistor `R3` to be removed from the netlist, and the schematic of the resulting circuit that **Xyce** simulates is shown in Fig. 14.9. Note that presence of the "`C`" in the `REMOVEUNUSED` statement does not cause **Xyce** to abort even though there are no capacitors in the netlist. Also, as in the case of a `REPLACEGROUND` statement, only one `.PREPROCESS REMOVEUNUSED` line may be present per netlist, or **Xyce** will abort.

A complete list of devices which can be removed via a `REMOVEUNUSED` statement is listed in Table 14.1. Note that in the case of MOSFETs and BJTs, three device terminals must be the same (the gate, source, and drain in the case of a MOSFET and the base, collector, and emitter in the case of a BJT) in order for an instance of either device to be removed from the netlist.

```
Circuit with improperly connected voltage source V2

V1 1 0 1
R1 1 2 1
R2 2 0 1
R3 2 2 1

.PREPROCESS REMOVEUNUSED R,C

.DC V1 1 1 0.1
.PRINT DC V(2)
.END
```

Figure 14.8. Circuit with an "unused" resistor R3 that gets removed from the netlist.

Keyword	Device Type
C	Capacitor
D	Diode
I	Independent Current Source

Keyword	Device Type
L	Inductor
M	MOSFET
Q	BJT
R	Resistor
V	Independent Voltage Source

Table 14.1: List of keywords and device types which can be used in a .PREPROCESS REMOVEUNUSED statement.

14.4 Adding Resistors to Dangling Nodes

Consider the netlist of Fig. 14.10 and the corresponding schematic of Fig. 14.11. Nodes 3 and 4 of the netlist are what we will henceforth refer to as *dangling nodes*. We say that node 4 dangles because it is only connected to the terminal of a single device, while we say that node 3 dangles because it has no DC path to ground. The first of these situations—connection to a single device terminal only—can arise, for example, in a netlist which contains nodes representing output pins that are not connected to a load device. For instance, the resistance R2 in Fig. 14.10 could represent the resistance of an output pin of a package that is meant to drive resistive loads. Hence, an actual physical implementation of the circuit of Fig. 14.11 would normally include a resistor between node 4 and ground, but, in creating the netlist, the presence of such an output load has been (either intentionally or unintentionally) left out.

The second situation—where a node has no DC path to ground—is sometimes an effect that is purposely incorporated into a design (e.g., the design of switched capacitor integrators—see, e.g., Chapter 10 of [17]), but it is also oftentimes the result of some form of error in the process of creating the netlist. For instance, when graphical user interfaces (GUIs) are used to create circuit schematics which are then translated into netlists via software, one very common unintentional error is to fail to connect two nodes which are intended to be connected. To illustrate this point, consider the schematic of Fig. 14.12. The schematic seems to indicate that the lower terminal of resistor R2 should be connected to node 3, but this is in fact not the case since there is a small gap between node 3 and the line which is intended to connect node 3 to the resistor. Such an error can often go unnoticed when creating a schematic of the netlist in a GUI. Thus, when the schematic is translated into a netlist file, the resulting netlist would *not* connect the resistor to node 3 and would instead create a new node at the bottom of the resistor, resulting in the circuit

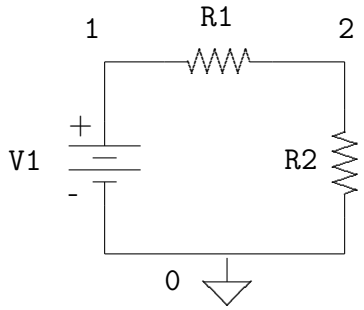

Figure 14.9. Circuit of Fig. 14.8 where the resistor R3 has been removed via the `.PREPROCESS REMOVEUNUSED` statement.

depicted in Fig. 14.11.

While neither of the above situations is necessarily threatening (the netlist of Fig. 14.10 will successfully run to completion in **Xyce**), there are times when it is desirable to somehow make a dangling node *not* dangle. For instance, returning to the example in which the resistor `R2` represents the resistance of an output pin, it may be the case that we wish to simulate the circuit when a 1K load is attached between node `4` and ground in Fig. 14.11. In the case where a node has no DC path to ground, the situation is slightly more dangerous if, for instance, the node in question is also connected to a high-gain device such as the gate of a MOSFET. Since the DC gate bias has a great impact on the DC current travelling through the drain and source of the transistor, not having a well-defined DC gate voltage can greatly degrade the simulated performance of the circuit.

In both of the prior examples, the only true way to "fix" each of these issues is to find all the dangling nodes in a particular netlist file and to augment the netlist at/near these nodes to obtain the desired behavior. If it is the case however, that the number of components in a circuit is very large (say on the order of hundreds of thousands of components), manually augmenting the netlist file for each dangling node becomes a practical impossibility if the number of such nodes is large. Hence, it is desirable for **Xyce** to be capabable of automatically augmenting netlist files so as to help remove dangling nodes from a given netlist. The command `.PREPROCESS ADDRESISTORS` is designed to do just this. Consider the netlist of Fig. 14.13. Assuming that this netlist is stored in the file `filename`, the `.PREPROCESS ADDRESISTORS` statements will cause **Xyce** to create a new netlist file called `filename_xyce.cir` that is depicted in Fig. 14.14. The line `.PREPROCESS ADDRESISTORS NODCPATH 1G` instructs **Xyce** to create a copy of the netlist file which contains a set of resistors of value 1G that are connected between ground and nodes which currently have

```
Circuit with two dangling nodes, nodes 3 and 4

V1 1 0 1
R1 1 2 1
C1 2 3 1
C2 3 0 1
R2 2 4 1

.DC V1 0 1 0.1
.PRINT DC V(2)
.END
```

Figure 14.10. Netlist of circuit with two dangling nodes, nodes 3 and 4.

no DC path to ground. Similarly, the line `.PREPROCESS ADDRESISTORS ONETERMINAL 1M` instructs **Xyce** to add to the same netlist file a set of resistors of value 1M which are connected between ground and devices that are connected to only one terminal. The resistor `RNODCPATH1` of Fig. 14.14 achieves the first of these goals while `RONETERM1` achieves the second. A schematic of the resulting circuit that is represented by the netlist in Fig. 14.14 is shown in Fig. 14.15

Some general comments regarding the use of `.PREPROCESS ADDRESISTOR` statements:

- **Xyce** does not terminate immediately after the netlist file is created. In other words, if **Xyce** is run on the netlist `filename` of Fig. 14.13, it will attempt to execute this netlist as given (i.e., it tries to simulate the circuit of Fig. 14.11) and generates the file `filename_xyce.cir` as a biproduct. It is important to point out that the resistors that are added at the bottom of the netlist file `filename_xyce.cir` do **not** get added to the original netlist when **Xyce** is running on the file `filename`. If one wishes to simulate **Xyce** with these resistors in place, one must run **Xyce** on `filename_xyce.cir` explicitly.

- The naming convention for resistors which connect to ground nodes which do not have a DC path to ground is `RNODCPATH<i>`, where `i` is an integer greater than 0; the naming convention is similar for nodes which are connected to only one device terminal (i.e., of the form `RONETERM<i>`). It should be noted that **Xyce** will not change this naming convention if a resistor with one of the above names already

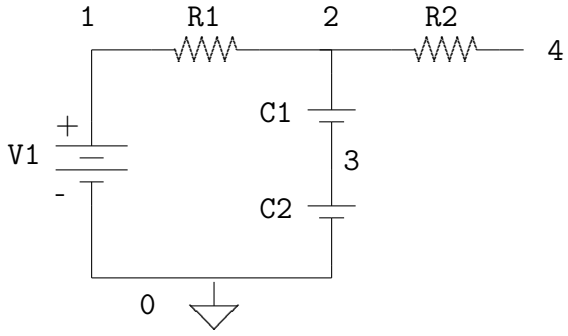

Figure 14.11. Schematic of netlist in Fig. 14.10.

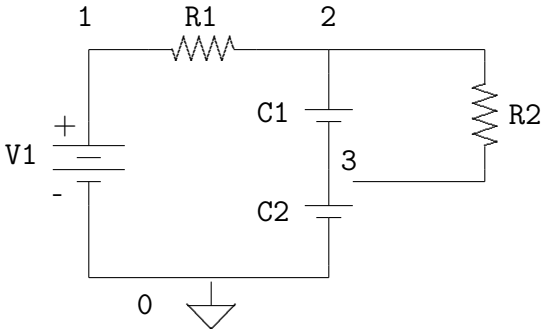

Figure 14.12. Schematic of a circuit with an incomplete connection between the resistor `R2` and node `3`.

exists in the netlist. Hence, if a resistor with the name `RNODCPATH1` exists in netlist file `filename`, and **Xyce** detects that there is a node in this netlist file that has no DC path to ground, **Xyce** will add *another* resistor with name `RNODCPATH1` to the netlist file `filename_xyce.cir` (assuming that either `.PREPROCESS ADDRESISTORS NODCPATH` or `.PREPROCESS ADDRESISTORS ONETERMINAL` are present in `filename`). If **Xyce** is subsequently run on `filename_xyce.cir`, it will exit in error due to the presence of two resistors with the same name.

- Both of the commands `.PREPROCESS ADDRESISTORS NODCPATH` and `.PREPROCESS ADDRESISTORS ONETERMINAL` do **not** have to be simultaneously present in a netlist file. The presence of either command will generate a file `filename_xyce.cir`, and the presence of both will not generate two separate files. As with other `.PREPROCESS`

```
Circuit with two dangling nodes, nodes 3 and 4

V1 1 0 1
R1 1 2 1
C1 2 3 1
C2 3 0 1
R2 2 4 1

.PREPROCESS ADDRESISTORS NODCPATH 1G
.PREPROCESS ADDRESISTORS ONETERMINAL 1M

.DC V1 0 1 0.1
.PRINT DC V(2)
.END
```

Figure 14.13. Netlist of circuit with two dangling nodes, nodes 3 and 4, with .PREPROCESS ADDRESISTORS statements.

commands, however, a netlist file is allowed to contain only one NODCPATH and one ONETERMINAL command each. If multiple NODCPATH and/or ONETERMINAL lines are found in a single netlist file, **Xyce** will exit in error.

- It is possible that a single node can both have no DC path to ground *and* be connected to only one device terminal. If both a NODCPATH and ONETERMINAL command are present in a given netlist file, **only** the resistor corresponding to the ONETERMINAL command is added to the netlist file filename_xyce.cir and the resistor corresponding to the NODCPATH command is omitted. If a NODCPATH command is present but a ONETERMINAL command is not, then a resistor corresponding to the NODCPATH command will be added to the netlist as usual.

- In generating the file filename_xyce.cir, the original .PREPROCESS ADDRESISTOR statements are commented out with a warning message. This is to prevent **Xyce** from creating the file filename_xyce.cir_xyce.cir when the file filename_xyce.cir is run. Note that this act is put in place simply to avoid generating redundant files. While filename_xyce.cir_xyce.cir would be slightly different from filename_xyce.cir (e.g., a different date and time stamp), both files would functionally implement the same netlist.

```
XYCE-generated Netlist file copy:  TIME='07:32:31 AM'
+ DATE='Dec 19, 2007'
*Original Netlist Title:

*Circuit with two dangling nodes, nodes 3 and 4


V1 1 0 1
R1 1 2 1
C1 2 3 1
C2 3 0 1
R2 2 4 1

*.PREPROCESS ADDRESISTORS NODCPATH 1G
*Xyce:  ".PREPROCESS ADDRESISTORS" statement
+ automatically commented out in netlist copy.
*.PREPROCESS ADDRESISTORS ONETERMINAL 1M
*Xyce:  ".PREPROCESS ADDRESISTORS" statement
+ automatically commented out in netlist copy.

.DC V1 0 1 0.1
.PRINT DC V(2)


*XYCE-GENERATED OUTPUT:  Adding resistors between ground
+ and nodes connected to only 1 device terminal:

RONETERM1 4 0 1M


*XYCE-GENERATED OUTPUT:  Adding resistors between ground
+ and nodes with no DC path to ground:

RNODCPATH1 3 0 1G

.END
```

Figure 14.14. Output file `filename_xyce.cir` which results from the `.PREPROCESS ADDRESISTOR` statements for the netlist of Fig. 14.12 (with assumed file name `filename`).

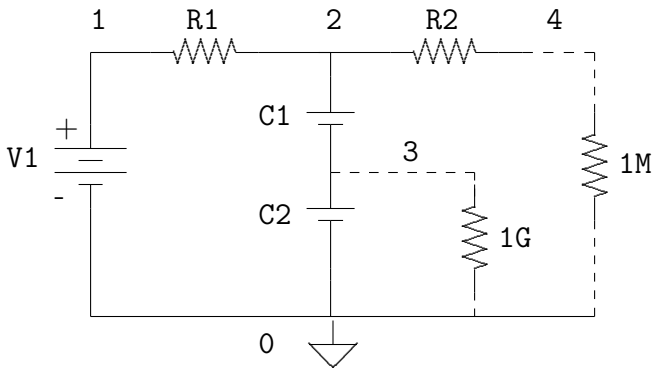

Figure 14.15. Schematic corresponding to the **Xyce**-generated netlist of Fig. 14.14.

15. TCAD (PDE Device) Simulation with **Xyce**

Chapter Overview

This chapter gives guidance on how to use the mesh-based device simulation capability of **Xyce**. It includes the following sections:

- Section 15.1, *Introduction*
- Section 15.2, *One Dimensional Example*
- Section 15.3, *Two Dimensional Example*
- Section 15.4, *Doping Profile*
- Section 15.5, *Electrodes*
- Section 15.6, *Meshing*
- Section 15.7, *Mobility Models*
- Section 15.8, *Bulk Materials*
- Section 15.9, *Solver Options*
- Section 15.10, *Output and Visualization*

15.1 Introduction

This chapter describes how to use the mesh-based device simulation functionality of **Xyce**. This capability is based on the solution a coupled set of partial differential equations (PDEs), discretized on a mesh such as the one in Figure 15.1. Such devices are often referred to as TCAD devices, where TCAD stands for Technology Computer Aided Design. While the rest of **Xyce** is intended to be similar to analog circuit simulators such as SPICE, the TCAD device capability is intended to be similar to well-known device simulators such as PISCES [18] and DaVinci [19].

Two different TCAD devices are available **Xyce**: a one-dimensional device and a two-dimensional device. These two devices have been implemented in a manner which allows them to be invoked in the same way as a conventional lumped parameter circuit device. Generally, this capability is intended for very detailed simulation of semiconductor devices, such as diodes, bipolar transistors, and MOSFETs. One possible application of this capability is the evaluation and/or analysis of conventional SPICE-style lumped parameter models.

NOTE: As of **Xyce** Release 2.1, the TCAD devices should still be considered to be a beta-level capability. The primary focus of **Xyce** has been traditional analog circuit simulation, so the TCAD devices have not been subject to the same level of testing as the traditional, SPICE-style devices. The TCAD (PDE) device simulator in **Xyce** should be regarded as a prototype for Charon, a high performance 3D device simulator that is under development at Sandia.

NOTE: The use of square brackets [] in the doping and electrode specifications is no longer correct as of **Xyce** Release 2.1. The correct delimiter to use for these parameters is now the curly bracket {}.

Equations

The equations of device simulation are described by many references including Kramer [20] and Selberherr [21]. The most common formulation, and the one that is used in **Xyce**, is the drift-diffusion (DD) formulation. This formulation consists of three coupled PDE's: a single Poisson equation for electrostatic potential and two continuity equations; one each for electrons and holes.

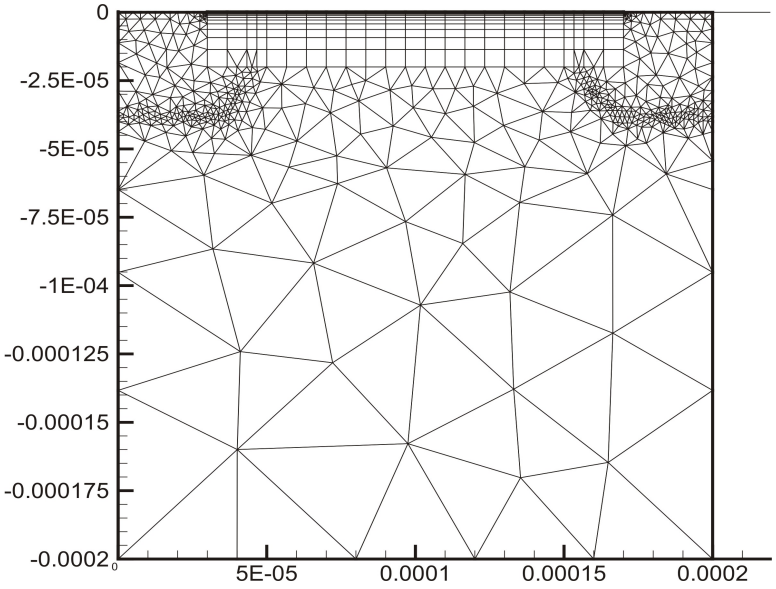

Figure 15.1. MOSFET Mesh Example.

Poisson equation

The electrostatic potential ϕ satisfies Poisson's equation:

$$-\nabla \cdot (\epsilon \nabla \phi(x)) = \rho(x) \tag{15.1}$$

where ρ is the charge density and ϵ is the permittivity of the material. For semiconductor devices, the charge density is determined by the local carrier densities and the local doping,

$$\rho(x) = q(p(x) - n(x) + C(x)) \tag{15.2}$$

Here $p(x)$ is the spatially-dependent concentration of holes, $n(x)$ the concentration of electrons, and q the magnitude of the charge on an electron. $C(x)$ is the total doping concentration, which can also be represented as $C(x) = N_D^+(x) - N_A^-(x)$, where N_D^+ the concentration of positively ionized donors, N_A^- the concentration of negatively ionized acceptors.

Species continuity equations

The continuity equations relate the convective derivative of the species concentrations to the creation and destruction of particles ("recombination/generation").

$$\frac{\partial n(x)}{\partial t} + \nabla \cdot \Gamma_n = -R(x) \quad (15.3)$$

$$\frac{\partial p(x)}{\partial t} + \nabla \cdot \Gamma_p = -R(x) \quad (15.4)$$

Here n is the electron concentration and p is the hole concentration. R is the recombination rate for both species. Γ_n and Γ_p are particle fluxes for electrons and holes, respectively. The sign of R is chosen because R is usually expressed as a recombination rate, and is positive if particles are annihilating. The right hand sides are equal since creation and destruction of carriers occurs in pairs.

One way in which the drift-diffusion model differs from other common formulations is the manner in which the quantities Γ_n and Γ_p are determined. The expressions used are:

$$\Gamma_n = n(x)\mu_n E(x) + D_n \nabla n(x) \quad (15.5)$$

$$\Gamma_p = p(x)\mu_p E(x) + D_p \nabla p(x) \quad (15.6)$$

Here μ_n, μ_p are mobilities for electrons and holes, and D_n, D_p are diffusion constants. $E(x)$ is the electric field, which is given by the gradient of the potential, or $-\partial\phi/\partial x$.

Discretization

Xyce uses a box-integration discretization, with the Scharfetter-Gummel method for modeling the flux of charged species. This method has been described in detail elsewhere [20] [21] [22], so it will not be described here.

15.2 One Dimensional Example

The one-dimensional device was the first PDE-based device to be implemented in **Xyce**. The single dimension limits its usefulness, but its simplicity makes it a good device to use for a preliminary example. One dimensional devices are almost always two-terminal diodes, and this fact allows for assumptions which simplify the specification and shorten the parameter list of the device.

An example netlist, for a simulation of a one-dimensional diode, is shown in Figure 15.2. The corresponding schematic is in Figure 15.3. The circuit is a regulator circuit, and is based on the principle that connecting one or more diodes in series with a resistor and a power supply will produce a relatively constant voltage. The input voltage (node 2) is a sinewave, with a frequency of 50 Hz and an amplitude of 1 V. The expected output (node 3) should be a (mostly) flat signal.

Netlist Explanation

In Figure 15.2, the PDE device instance line is in red, while the PDE device model line is in blue. Currently, there are almost no model parameters for PDE devices. The model line serves only to set the level. The default level is 1, for a one-dimensional device. Two dimensional devices are invoked by setting `level=2`. Note that in this example, the level is not explicitly set, and so the default (1) is used.

The instance line is where most of the specific parameters are set for a TCAD device. In this example, the line appears as:

```
Z1 3 4 DIODE na=1.0e19 nd=1.0e19 graded=0 l=5.0e-4 nx=101
```

`na` and `nd` are doping parameters, and represent the majority carrier doping levels on the N-side and the P-side of the junction, respectively. `graded=0` is also a doping parameter, and specifies that the junction is not a graded junction, but is an abrupt step-function junction instead. `l=5.0e-4` specifies the length of the device, in cm. `nx=101` specifies that there are 101 mesh points, including the two endpoints. For the one-dimensional device, the mesh is always uniform, so the size of each mesh cell, Δx will be:

$$\Delta x = \frac{l}{nx-1} = \frac{\text{5.0e-4 cm}}{100} = \text{5.0e-6 cm} \tag{15.7}$$

The mesh points $i = 0 - 101$ will have the following locations, x_i:

$$x_i \quad = \quad i\Delta x$$

```
PDE Diode Regulator Circuit
VP 1 0 PULSE(0 5 0.0 2.0e-2 0.0 1.0e+20 1.2e+20)
VF 2 1 SIN(0 1 50 2.0e-2)
VT1 4 0 0V
R1 2 3 1k

 * TCAD/PDE Device
Z1 3 4 DIODE na=1.0e19 nd=1.0e19 graded=0
+ l=5.0e-4 nx=101

 .MODEL DIODE  ZOD

.TRAN 1.0e-3 12.0e-2
.print TRAN format=tecplot
+ v(1) v(2) v(3) v(4) I(VF) I(VT1)

.options NONLIN maxstep=100 maxsearchstep=3
+ searchmethod=2  nox=0

.options TIMEINT reltol=1.0e-3 abstol=1.0e-6
+ resettrannls=0

.END
```

Figure 15.2. One dimensional diode netlist. This circuit is a voltage regulator. The input signal should be a sinewave, while the output signal should be nearly flat. For the result of this netlist, see Figure 15.4

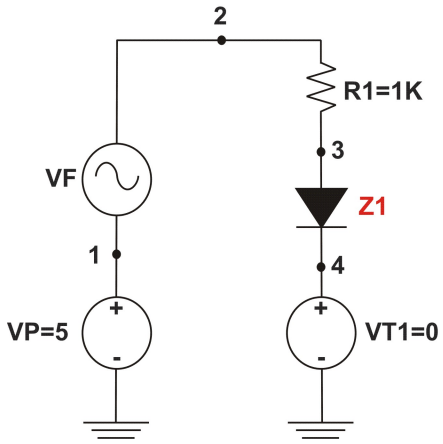

Figure 15.3. Voltage regulator schematic. The diode, Z1, is the PDE device in this example.

$$
\begin{aligned}
x_o &= \text{0.0 cm} \\
x_1 &= \text{5.0e-6 cm} \\
&\vdots \\
x_{101} &= \text{5.0e-4 cm}
\end{aligned}
$$

Boundary Conditions and Doping Profile

Note that nothing has been specified in the example netlist about electrodes, or boundary conditions, and that the doping specification is minimal. This is because the example relies a lot on default parameters. A one dimensional device can only have exactly 2 electrodes connected to the circuit. These two electrodes are at opposite ends of the domain, one at the first mesh point (`x=0.0 cm`, `i=0`) and the other at the opposite end of the domain, at the last mesh point (`x=5.0e-4 cm`, `i=101`).

The electrode associated with the first mesh point (`x=0.0 cm`) is connected to the *second* circuit node on the instance line, while the electrode associated with the last mesh point (`x=l`) is connected to the *first* circuit node on the instance line. For the doping used in this example, the junction is in the exact center of the device (`x=l/2`), and the n-side is

the region defined by `x<l/2`, and the p-side is the region defined by `x>l/2`. This default doping, along with the electrode-circuit connectivity, result in the one-dimensional device to behave like a traditional SPICE-style diode. For a complete discussion of how to specify a doping profile see section 15.4. For a complete discussion of how to specify electrodes in detail (including boundary conditions), see section 15.5.

Results

The transient result of this circuit is shown in Figure 15.4. The input signal (node 2) is represented by the blue line, and the output signal (node 3) is represented by the red line. The voltage drop across the diode is nearly the same for a wide range of currents, and is approximately 0.67 V. The voltage drop across the series resistor, `R1`, is much more sensitive to the current magnitude, and so most of the voltage variation of the input sinewave is accounted for by `R1`.

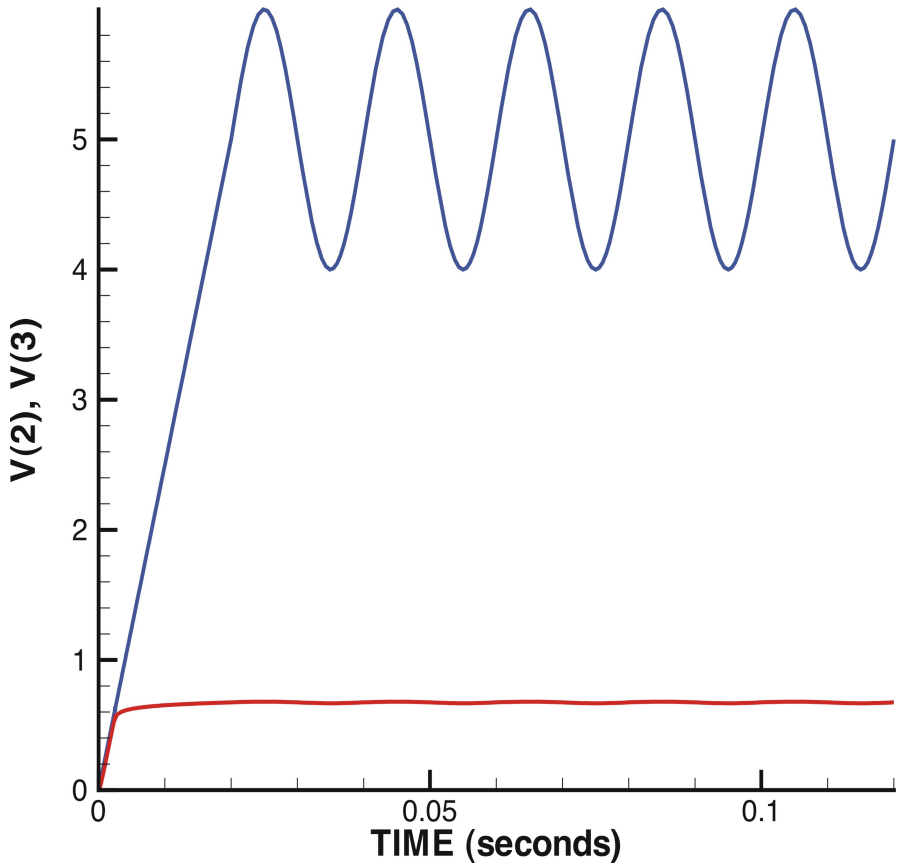

Figure 15.4. Transient result for the voltage regulator circuit in Figure 15.2. The input voltage is represented by the blue line, while the output voltage is given by the red line.

15.3 Two Dimensional Example

An example netlist, for a simulation of a two-dimensional bipolar transistor, is shown in Figure 15.5. As before, the PDE device instance line is in red, while the PDE device model line is in blue. In this case, note that the level has been specified on the model line, and it has been set to 2. This is required for the two-dimensional device. This particular example is a DC sweep of a bipolar transistor device. A schematic, illustrating this circuit is shown in Figure 15.6.

Netlist Explanation

The two-dimensional device can have 2-4 electrodes. (this limitation will be relaxed in future versions of **Xyce**) In this example there are three; node 5, node 3 and node 7. These correspond to the three names on the "node" line, which appears as:

```
+ node = {name = collector, base, emitter}
```

This line specifies that node 5 is connected to an electrode named "collector", node 3 is connected to an electrode named "base", and node 7 is connected to an electrode named "emitter". Although this example only contains the electrode names, the "node" specification can contains a lot of information. For a full explanation of all the electrode parameters, see section 15.5.

The next line contains parameters concerned with plotting the results, and appears as follows:

```
+ tecplotlevel=2 txtdatalevel=1
```

Note that these are not related to the output specified by `.PRINT`, which outputs circuit data. The `tecplotlevel` command enables files to be output which are readable by tecplot. Tecplot can then be used to create contour plots of quantities such as the electron density, electrostatic potential and the doping profile. Figures 15.7 and 15.8 contain examples of tecplot-generated contour plots, which were generated from the results of this example.

The `txtdatalevel` command enables a text file with volume averaged information to be output to a file. Currently, both of these output files will be updated at each time step or DC sweep step.

The next line, `mobmodel=arora`, specifies which mobility model to use. For more detail on available mobility models, see section 15.7.

```
Two Dimensional Example
VPOS  1 0 DC 5V
VBB   6 0 DC -2V
RE    1 2 2K
RB    3 4 190K

 Z1BJT 5 3 7 PDEBJT meshfile=internal.msh
+ node = {name = collector, base, emitter}
+ tecplotlevel=2 txtdatalevel=1
+ mobmodel=arora
+ l=2.0e-3  w=1.0e-3
+ nx=30     ny=15

* Zero volt sources acting as an ammeter to measure the
* base, collector, and emmitter currents, respectively
VMON1 4 6 0
VMON2 5 0 0
VMON3 2 7 0

.MODEL PDEBJT   ZOD  level=2

.DC VPOS 0.0 12.0 0.5 VBB -2.0 -2.0 1.0
.options LINSOL type=superlu
.options NONLIN maxstep=70 maxsearchstep=1
+ searchmethod=2 in_forcing=0 nlstrategy=0

.options TIMEINT reltol=1.0e-3 abstol=1.0e-6
+ firstdcopstep=0 lastdcopstep=1

.PRINT DC V(1) I(VMON1) I(VMON2) I(VMON3)

.END
```

Figure 15.5. Two-dimensional BJT netlist. Some results of this netlist can be found in Figures 15.7 and 15.8.

The last two lines, specify the mesh of the device, and are given by:

```
+ l=2.0e-3 w=1.0e-3
+ nx=30 ny=15
```

This numbers are used in nearly the same way as the `l` and `nx` parameters were used in the one-dimensional case. The mesh is Cartesian, and the spacing is uniform.

Doping Profile

As in the one-dimensional example, the two-dimensional example in figure 15.5 does not specify anything about the doping profile, and thus relies upon defaults. In this case there are three specified electrodes, which by default results in the doping profile of the bipolar junction transistor (BJT). For a complete description of how to specify a doping profile in detail, see section 15.4. This section also describes the various default impurity profiles.

Boundary Conditions and Electrode Configuration

As in the one-dimensional example, the two-dimensional example in figure 15.5 does not specify anything about the electrode configuration or the boundary conditions, and relies on default settings. To be consistent with the default 3-terminal doping, the device has terminals that correspond to that of a BJT. All three electrodes (collector, base, emitter) are along the top of the device.

By default all electrodes are considered to be neutral contacts. The boundary conditions applied to the electron density, hole density and electrostatic potential are all Dirichlet conditions.

For a complete discussion of how to specify electrodes in detail (including boundary conditions), see section 15.5.

Results

Results for the two-dimensional example can be found in Figures 15.7, 15.8 and 15.9. The first two figures are contour plots of the electrostatic potential. The first one corresponds to the first DC sweep step, where VPOS is set to 0.0 Volts. The second one corresponds to the final DC sweep step, in which VPOS has a value of 12.0 volts. The voltage source

VPOS applies a voltage to the emitter load resistor, RE, so some of the 12.0V is dropped across RE, an the rest is applied to the BJT.

The third figure is an I-V curve of the dependence of the three terminal currents on applied emitter voltage. For the entire sweep, a negative voltage of 2.0 V has been applied to the base load resistor, and as this transistor is a PNP transistor, this results in the transistor being in an "on" state. The emitter-collector current varies nearly linearly with the applied emitter voltage. Also, the three currents sum to nearly zero, which one would expect because of current conservation.

Note that the mesh is visible in Figure 15.7, and was generated using the internal "uniform mesh" option. Generally using this sort of mesh will work numerically, in that **Xyce** will converge to an answer. However, this mesh will probably not produce a very accurate result, as it does not resolve the depletion regions very well. In order to obtain better accuracy, either a finer uniform mesh would need to be used, or a nonuniform mesh, refined in around the depletion regions should be used. As described in section 15.6, refined, nonuniform meshes must be read in from an external mesh generator, such as the SGFramework [20].

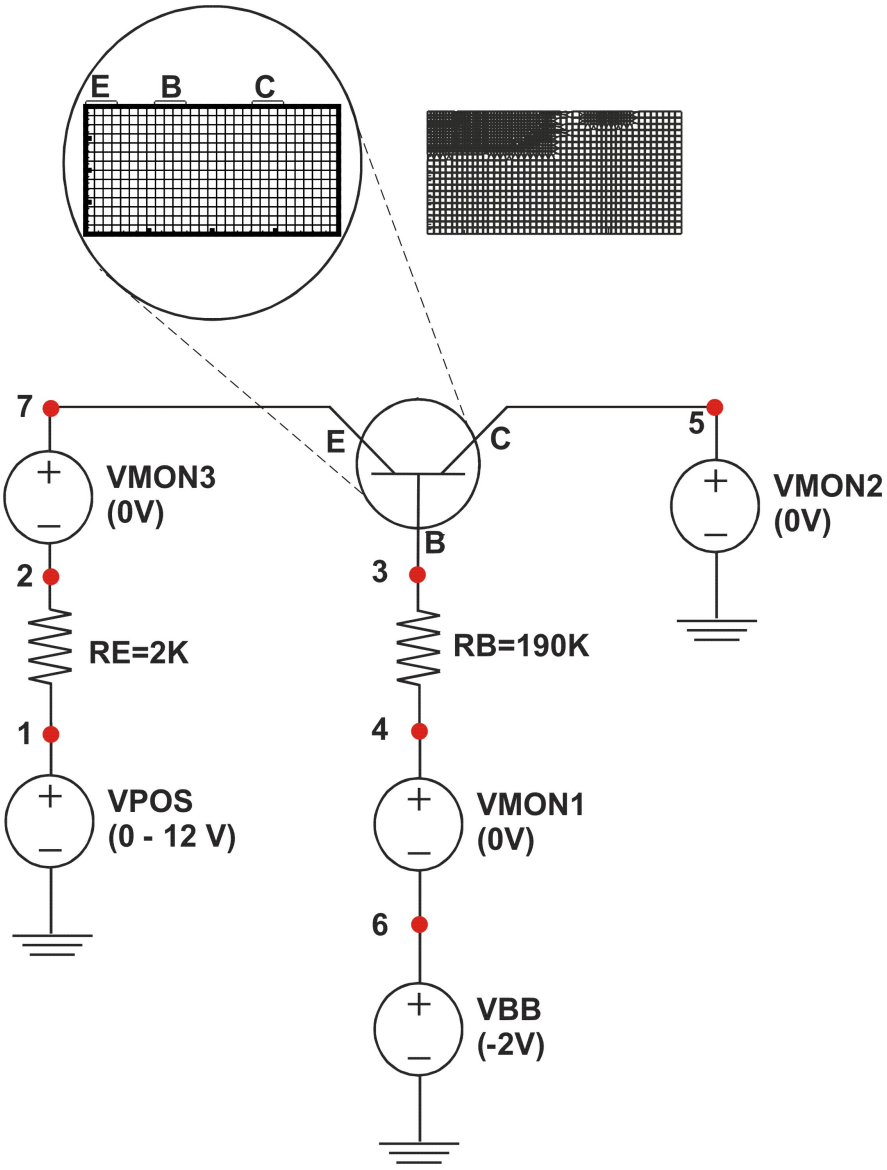

Figure 15.6. Two-Dimensional BJT Circuit Schematic. This schematic is for the circuit described by the netlist in Figure 15.5. The mesh in the large circle is the mesh used in the example. The other mesh, which contains some mesh refinement, is included in the figure as an example of what is possible with an external mesh generator.

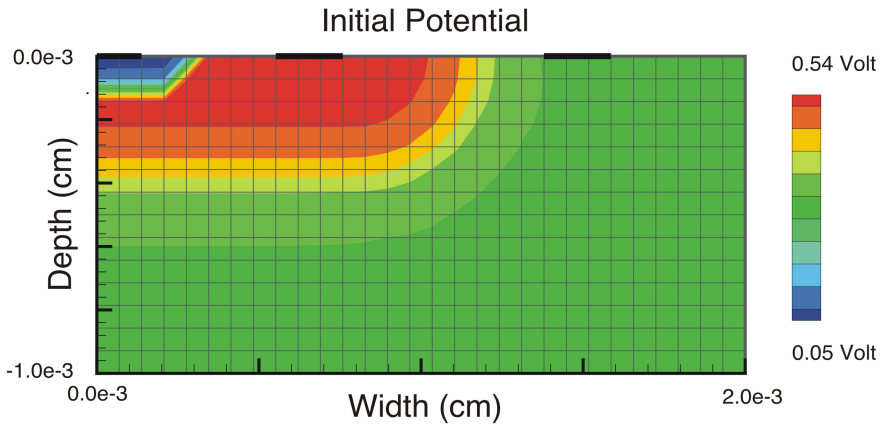

Figure 15.7. Initial Two-Dimensional BJT Result. Contour plot of the electrostatic potential at the first DC sweep step of the netlist in Figure 15.5. Note the mesh, which was generated using the internal "uniform mesh" option. This plot was generated using Tecplot.

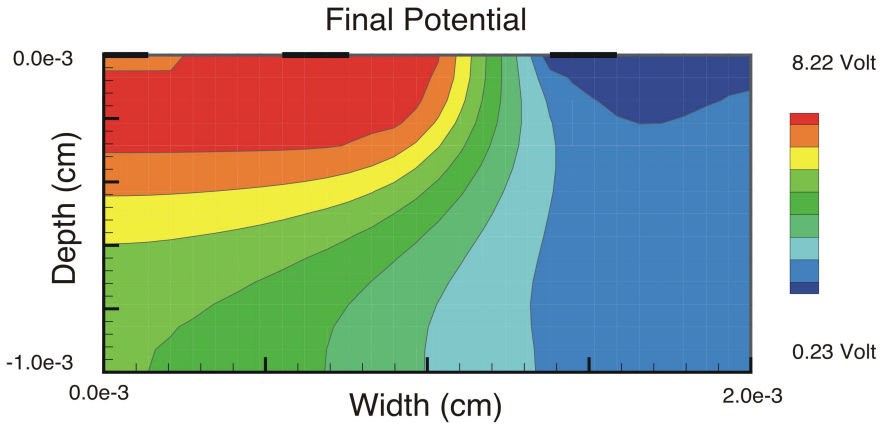

Figure 15.8. Final Two-Dimensional BJT Result. Contour plot of the electrostatic potential at the last DC sweep step of the netlist in Figure 15.5. This plot was generated using Tecplot.

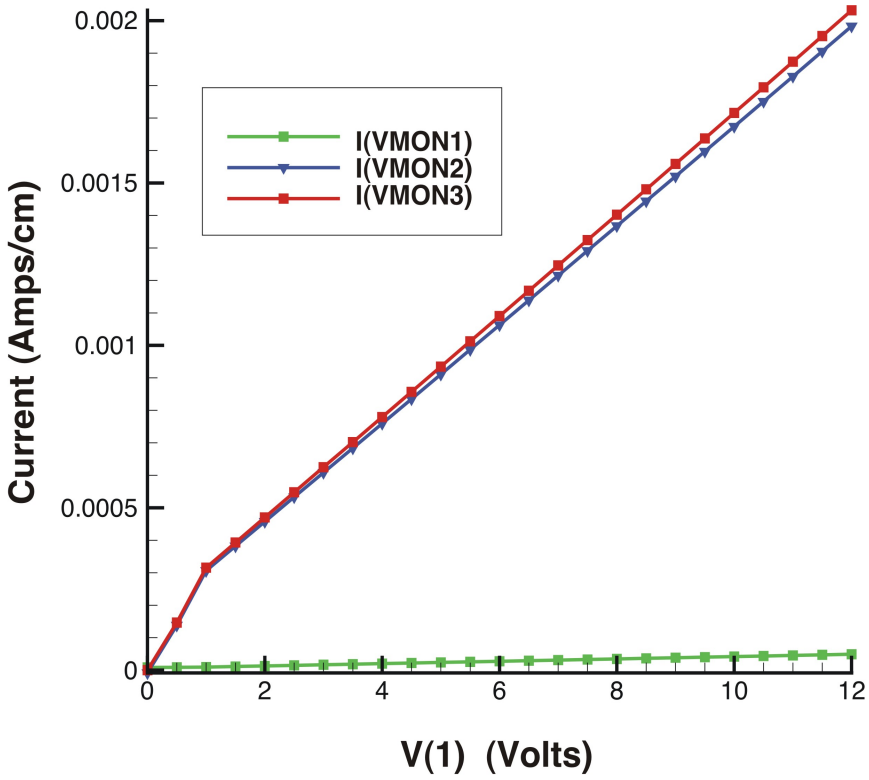

Figure 15.9. I-V Two-Dimensional BJT Result, for the netlist in Figure 15.5. The x-axis is in Volts. The three plotted currents are through the three BJT electrodes, and as expected they add (if corrected for sign) to zero. I(VMON1) is the base current, I(VMON2) the collector current, and I(VMON3) the emitter current. V(1) is the voltage applied to the emitter load resistor, RE. This plot was generated using Tecplot.

15.4 Doping Profile

In the two examples, no doping parameters were specified, and **Xyce** used the defaults. Default profiles are uniquely specified by the number of electrodes. In practice, especially for two-dimensional simulations, the user will generally need to specify the doping profile manually.

NOTE: If an external mesh (from the SGFramework) is used, the doping profile will be read in from the mesh file, and it is not necessary (or appropriate) to specify any doping in the netlist.

Manually Specifying the Doping

A circuit netlist, which includes a one-dimensional device with a detailed, manual specification of the doping profile, is given in figure 15.10. A similar, two-dimensional, version of this problem is given in figure 15.12. For the purposes of this discussion, the one-dimensional example will be referred to, but information conveyed is equally applicable to the two-dimensional case.

In both examples, the parameters associated with doping are in red font. The doping is specified with one or more regions, which are summed together to get the total profile. Doping regions are specified in a tabular format, with each column representing a different region.

In the one-dimensional example, there are three regions, which are illustrated in figure 15.11. Region 1 is a uniform n-type doping, with a constant magnitude of 4.0e+12 donors per cubic cm. This magnitude is set by the parameter nmax. As the doping in this region is spatially uniform, the only meaningful parameters are `function` (which in this case specifies a spatially uniform distribution), `type` (ntype or ptype) and `nmax`. The others (nmin through flaty) are ignored for a spatially uniform region.

Region 2 is a more complicated region, in that the profile varies with space. This region is doped with p-type impurities, and has a Gaussian shape. Semiconductor processing often consists of an implant followed by an anneal, which results in a diffusive profile. The Gaussian function is a solution to the diffusion problem, when it is assumed that the impurity exists in a fixed quantity. Thus, the Gaussian shape is an appropriate choice for the doping regions of a lot of devices.

The peak of the Region 2 doping profile is given by the parameter nmin, and is 1.0e+19

```
Doping and Electrode specification example
TITLE     Xyce PN Junction Simulation
vscope   0   1   0.0
rscope   2   1   50.0
cid      3   0   1.0u
r1       4   3   1515.0
vid      4   0   5.00
Z1DIODE 2 3 PDEDIODE nx=301 l=26.0e-4
* DOPING REGIONS:    region 1,  region 2,   region 3
+ region= {function = uniform,  gaussian,  gaussian
+          type     =   ntype,     ptype,     ntype
+          nmax     = 4.0e+12,   1.0e+19,   1.0e+18
+          nmin     = 0.0e+00,   4.0e+12,   4.0e+12
+          xloc     =    0.0 ,  24.5e-04,   9.0e-04
+          xwidth   =    0.0 ,   4.5e-04,   8.0e-04
+          flatx    =    0   ,        0 ,       -1 }
*--------end of  Diode PDE device ----------------
.MODEL PDEDIODE  ZOD  level=1
.options LINSOL type=superlu
.options NONLIN maxsearchstep=1 searchmethod=2
.options TIMEINT reltol=1.0e-3 abstol=1.0e-6
.DC vscope 0 0 1
.print DC v(1) v(2) v(3) v(4) I(vscope) I(vid)
.END
```

Figure 15.10. One-dimensional example, with detailed doping.

acceptors per cubic cm. This peak has a location in the device which is specified by `xloc=24.5e-4 cm`. The parameters nmin and xwidth are fitting parameters.

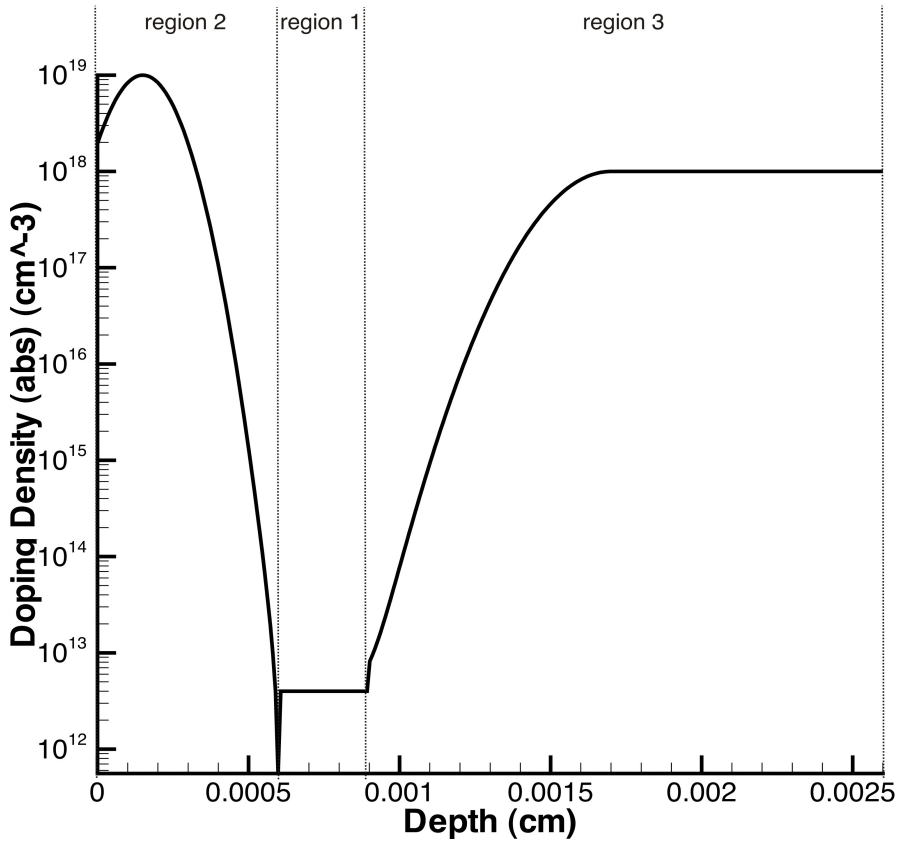

Figure 15.11. Doping Profile, Absolute Value. This corresponds to the doping specified by the netlist in figure 15.10

Region 3 is also based on a Gaussian function, but unlike Region 2, it is flat on one side of the peak. This is set by the `flatx` parameter. The "flat" parameters follow the convention given by Table 15.1.

flatx or flaty value	Description	1D Cross Section
0	Gaussian on both sides of the peak (`xloc`) location.	
+1	Gaussian if `x>xloc`, flat (constant at the peak value) if `x<xloc`.	
-1	Gaussian if `x<xloc`, flat (constant at the peak value) if `x>xloc`.	

Table 15.1: Description of the flatx, flaty doping parameters

Default Doping Profiles

Xyce has a few default doping profiles which are invoked if the user doesn't bother to specify detailed doping information. The default doping profiles are an artifact of early TCAD device development in **Xyce**, but are sometimes still useful. In particular, the simple step-junction diode is often a useful canonical problem. It is convenient to invoke a step junction doping without having to use the more complex region tabular specification.

Most real devices will have doping profiles that do not exactly match the default profiles. When attempting to simulate a realistic device, it will be necessary to skip the defaults and use the region tables described in the previous section.

One Dimensional Case

For the one-dimensional case, it is assumed that the doping profile is that of a simple junction diode, with the junction location exactly in the middle. The acceptor and donor concentrations are given by the parameters `Na` and `Nd`, respectively.

Note that the usage of `Na` and `Nd`, implicitly specifies a step junction doping profile, and is mutually exclusive with the more complex "doping region" table specification, described in section 15.4. If a netlist is input to **Xyce** which includes both a region table and `Na` (or `Nd`), the code will immediately exit with an error.

Two Dimensional Case

Doping level defaults in the two dimensional case are somewhat more complicated than in the one-dimensional case, because having two-dimensions allows for more configurations, and an arbitrary number (2-4) of electrodes. In **Xyce**, it was decided that the default doping profiles would be determined uniquely by the number of electrodes. The three available default dopings are given in Table 15.2. In the case of the BJT and MOSFET dopings, it is possible to specify either n-type or p-type using the `type` instance parameter. If the detailed, manual doping is used, then the `type` parameter is ignored.

For a two-electrode device, the default doping is that of a simple diode. The acceptor and donor doping parameters, `Na` and `Nd` are used in the same manner as in the one-dimensional device. As in the one-dimensional device, the junction is assumed to be exactly in the middle of the domain.

For a three-electrode device (like the example), the default doping is that of a bipolar junction transistor (BJT). By default the transistor is a PNP, but by setting the instance parameter `type=NPN`, an NPN transistor can be specified instead. The two-dimensional example in section 15.3 relies on this default.

For a four-terminal device, the default doping is that of a metal-oxide-semiconductor (MOSFET). Currently, the maximum number of electrodes is four, and no default profiles are available for more than four electrodes. By default this transistor is assumed to be NMOS, rather than PMOS.

Number of Electrodes	Doping Profile
2	Step Function Diode
3	Bipolar Junction Transistor (BJT)
4	Metal-Oxide Semiconductor Field-Effect Transistor(MOSFET)

Table 15.2: Default Doping profiles for different numbers of electrodes

15.5 Electrodes

In the two examples, minimal electrode were specified, and **Xyce** used the defaults. In practice, especially for two-dimensional simulations, the user will need to specify the electrodes in more detail.

NOTE: If an external mesh (from the SGFramework) is used, some of the electrode information (the locations, and lengths) will be specified in the mesh file, so they should not be specified in the netlist.

Manually Specifying the Electrodes

A detailed electrode specification is specified in blue font in Figure 15.12. As with the doping parameters, the electrode parameters are specified in a tabular format, in which each columns of the table specifies the parameters for a different electrode. The most important parameter (for getting the code to run without immediately exiting with an error) is the `name` parameter. It is the only required parameter.

The number of specified electrodes must match the number of connected circuit nodes, and the order of the electrode columns, from left to right, is in the same order as the circuit nodes, also from left to right. In the example of Figure 15.12, the first electrode column, which specifies an electrode named "anode", is connected to the circuit through circuit node 2. Respectively, the second column, for the "cathode" electrode, is connected to the circuit by circuit node 3.

If using an external mesh (see section 15.6), the external mesh file must have this same number of electrodes as well. Also, if using the external mesh, the electrode names specified in the electrode table must match (case insensitive) with the electrode names used by the external mesh.

Boundary Conditions

In the example, the `bc` parameter has been set to "Dirichlet" on all the electrodes, which is the default. The `bc` parameter sets the type of boundary condition that is applied to the density variables, the electron density and the hole density. There are two possible settings for the `bc` parameter, Dirichlet and Neumann. If Dirichlet is specified, the electron and hole densities are set to a specific value at the contact, and the applied values enforce charge neutrality. See the **Xyce** Reference Guide for the charge-neutral equation [3]. If Neumann

```
Doping and Electrode specification example
vscope   1   0   0.0
rscope   2   1   50.0
cid      3   0   1.0u
r1       4   3   1515.0
vid      4   0   1.00
*------------ Diode PDE device -----------------
Z1DIODE 2 3 PDEDIODE
+ tecplotlevel=1 txtdatalevel=1 cyl=1
+ meshfile=internal.msh
+ nx=25  l=70.0e-4 ny=40  w=26.0e-4
 * ELECTRODES:              ckt node 2, ckt node 3
+ node = {name            =     anode,    cathode
+         bc              = dirichlet,  dirichlet
+         start           =       0.0,        0.0
+         end             =   70.0e-4,    70.0e-4
+         side            =       top,     bottom
+         material        =   neutral,    neutral
+         oxideBndryFlag =         0,          0 }
* DOPING REGIONS:    region 1,  region 2,   region 3
+ region= {function = uniform,  gaussian,  gaussian
+          type     =   ntype,     ptype,     ntype
+          nmax     = 4.0e+12,   1.0e+19,   1.0e+18
+          nmin     = 0.0e+00,   4.0e+12,   4.0e+12
+          xloc     =    0.0 ,  60.0e-04,    100.0
+          xwidth   =    0.0 ,   4.0e-04,      1.0
+          yloc     =    0.0 ,  24.5e-04,   9.0e-04
+          ywidth   =    0.0 ,   4.5e-04,   8.0e-04
+          flatx    =    0   ,       -1 ,       -1
+          flaty    =    0   ,        0 ,       -1 }
*--------end of  Diode PDE device ----------------
.MODEL PDEDIODE  ZOD  level=2
.options LINSOL type=superlu
.options NONLIN maxsearchstep=1 searchmethod=2
.options TIMEINT reltol=1.0e-3 abstol=1.0e-6
.DC vscope 0 0 1
.print DC v(1) v(2) v(3) v(4) I(vscope) I(vid)
.END
```

Figure 15.12. Two-dimensional example, with detailed doping and detailed electrodes.

is specified, a zero-flux condition is applied, which enforces that the current through the electrode will be zero.

This parameter does not affect the electrostatic potential boundary condition. The boundary condition applied to the potential is always Dirichlet, and is (in part) determined from the connected nodal voltage. To apply a specific voltage to an electrode contact, a voltage source should be attached to it, such as `VBB` in the schematic Figure 15.6.

Electrode Material

Several different electrode materials can be specified. A list is given in Table 15.3. The main effect of any metal (non-neutral) material is the impose a Schottky barrier at the contact. This generally makes numerical solution more difficult, so any materials should be applied with caution.

The **Xyce** Reference Guide [3] has a detailed description of Schottky barriers and how they are imposed on contacts in **Xyce**. Also, values for electron affinities of various bulk materials and workfunction values for the various metal contacts are given in the Reference Guide.

Material	Symbol	Comments
neutral	neutral	Default
aluminum	al	
p+-polysilicon	ppoly	
n+-polysilicon	npoly	
molybdenum	mo	
tungsten	w	
molybdenum disilicide	modi	
tungsten disilicide	wdi	
copper	cu	
platinum	pt	
gold	au	

Table 15.3: Electrode Material Options. Neutral contacts are the default, and pose the least problem to the solvers.

There is also an `oxideBndryFlag` parameter, which if set to true (`1`), will model the contact

as having an oxide layer in between the metal contact and the bulk semiconductor. Note that this oxide layer model does not currently include displacement current, so transient capacitive effects will not be seen in the results.

Location Parameters

Each electrode has three location parameters: `start`, `end`, and `side`. These are only necessary if using the internal mesh and should not be specified if using an external, SGFramework mesh.

For the internal mesh, the mesh is assumed to be rectangular, and any electrode is assumed to be on one of the four sides. The four side possibilities are: `top`, `bottom`, `right` and `left`. These four sides are parallel to mesh directions. The `start` and `end` parameters are floating point numbers which specify the starting and ending location of an electrode, in units of cm.

The lower left hand corner of the mesh rectangle is located at the origin. A `side=bottom` electrode with `start=0.0` and `end=1.0e-4` will originate at the lower left hand corner of the mesh (x=0.0, y=0.0) and end at (x=1.0e-4, y=0.0).

NOTE: Xyce will attempt to match the specified electrode to the specified mesh. However, if the user specifies a mesh that is not consistent with the electrode locations, the electrodes will not be able to have the exact length specified. For example, if the mesh spacing is $\Delta x =$ 1.0e-5, then the electrodes can only have a length that is a multiple of 1.0e-5.

Electrode Defaults

There are defaults for all the electrode parameters except the names. In practice, the locations of the electrodes will usually be explicitly specified (either using the electrode table, or as part of an external mesh file). Default electrode locations have been created to correspond with the default dopings, and they should only be used in that context.

Location Parameters

In practice, the locations of the electrodes will usually be explicitly specified, but they have defaults to correspond with the default dopings. The default electrode locations in one-dimensional devices are for that of a diode. One electrode is located at `x=xmin`, while the other is located at `x=xmax`.

The default electrode locations in two-dimensional devices are dependent on the number of electrodes, similar to the default dopings. Table 15.2 can be used to determine the configurations. For the two-terminal diode, the two electrodes are along the y-axis, at the `x=xmin` and `x=xmax` extrema. For the three-terminal BJT , all three electrodes are parallel to the x-axis, along the top, at `y=ymax`. For the four-terminal MOSFET, the drain, gate, and source electrodes are also along the top, but the bulk electrode spans the entire length of the bottom of the mesh, at `y=ymin`.

Other Parameters

The default contact material is `neutral`. The default `oxideBndryFlag` is false (`0`). The default boundary condition (`bc`) is `Dirichlet`.

15.6 Meshes

Meshes from the SG Framework (External, 2D)

It is possible to have **Xyce** read in a two-dimensional mesh which was generated externally, by the SGFramework [20]. The mesh pictured in Figure 15.1 is such a mesh, and so is the refined mesh (not inside the circle) in Figure 15.6. To use an SGF-generated mesh, the instance parameter, "meshfile" must be used, and set to be the name of the SGFramework-generated file. **Xyce** will assume that the mesh file is located in the local execution directory. One advantage of using an externally generated mesh (over an internally generated mesh - see next section) is that external meshing tools are more sophisticated, and in particular have mesh refinement capabilities.

Instructions for the usage of the SGFramework is outside the scope of this document. If the user wishes to generate meshes in this manner, it is best to consult Kramer [20]. Future versions of **Xyce** may accept mesh files generated by other mesh generators, such as Cubit [23].

Cartesian Meshes (Internal, 1D and 2D)

One dimensional and two-dimensional devices can both create Cartesian meshes, without requiring an external mesh generator. For the two-dimensional devices, it is necessary to specify `meshfile=internal.msh` to invoke the Cartesian meshing capability. For one-dimensional devices, this isn't needed, as there is no other option.

Meshes generated in this manner are very simple, in that there are only two parameters per dimension, and the resulting mesh is uniform. An example of such a mesh can be seen in Figure 15.7. The mesh spacing is determined from the following expressions:

$$\Delta x = \frac{l}{nx - 1} \tag{15.8}$$

$$\Delta y = \frac{w}{ny - 1} \tag{15.9}$$

This mesh specification assumes that the domain is a rectangle. Non-rectangular domains can only be described using an external mesh program.

Cylindrical meshes, 2D

For two-dimensional devices, the simulation area may be a cylinder slice. This capability is turned on by the instance parameter, `cyl=1`. For an example, see Figure 15.13. It is assumed that the axis of the cylinder corresponds to the minimum radius (or x-axis value) of the mesh, while the circumference corresponds to the maximum radius (or maximum x-axis value). This feature can be applied to either external or internal two-dimensional meshes.

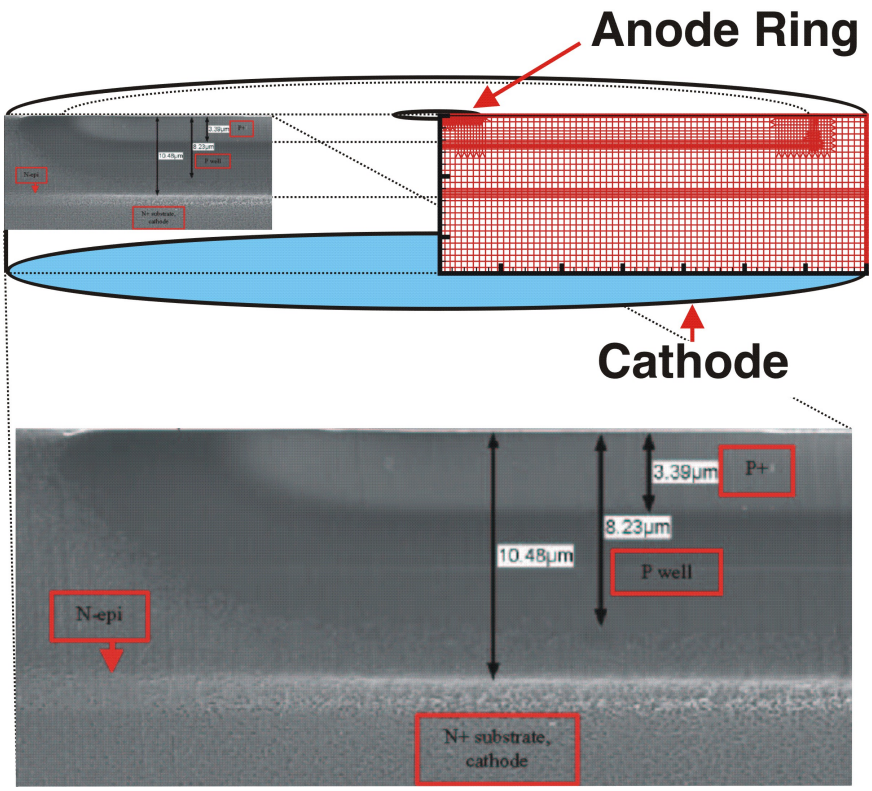

Figure 15.13. Cylindrical Mesh Example. This mesh has been designed to match the electron microscope image, which is of a stockpile device.

15.7 Mobility Models

There are several mobility models available to both the one and two dimensional devices, and they are listed in Table 15.4. These models are fairly common, and can be found in most device simulators. [18] [19] These models are described in more detail in the **Xyce** Reference Guide [3].

Mobility Name	Description	Reference
arora	Basic mobility model	Arora, et al. [24]
analytic	Basic mobility model	Caughy and Thomas [25]
carr	Includes carrier-carrier interactions	Dorkel and Leturq [26]

Table 15.4: Mobility models available for PDE devices

Specifying the mobility model from the netlist is done by setting the `mobmodel` parameter to the name of the model. Model names are given in the first column of Table 15.4. The mobility model is specified as an instance parameter on the device instance line, as (typically) `mobmodel=arora`. See the usage in Figure 15.5 for a more detailed example.

The default mobility is the ”carr” mobility, which includes carrier-carrier interactions. This model has a stronger dependence on carrier density than the other two models, and introduces some nonlinearity into the problem. If having convergence problems, consider using either the ”arora” or ”analytic” model, as both of these models are a little bit simpler.

15.8 Bulk Materials

The bulk material is specified using the `bulkmaterial` instance parameter. **Xyce** currently supports Silicon (`si`) as a bulk material and this is the default. It can also simulate Gallium Arsenide (`gaas`) and Germanium (`ge`), but these materials have not been extensively tested.

The mobility models described in the previous section each support all three materials, and the dielectric permittivity is correct for all three, but the carrier lifetime models may not be. These issues will be resolved in a future **Xyce** release.

15.9 Solver Options

Problems that are based on TCAD/PDE devices have different optimal solver settings than do analog circuit problems. Generally, as these devices are mesh-based, and have a more predictable topology, iterative linear solvers have a better chance of being successful than they do for analog circuit simulation. Note that if using a direct solver (klu, superlu, or ksparse), the best option is superlu, because ksparse and klu have both been optimized for circuits (not mesh-based PDE problems), while superlu is more general.

On the nonlinear solver level, voltage limiting doesn't have an obvious application to PDE devices, and quadratic line search appears to be the best algorithm. The solver options specified in the example netlist Figure 15.5 are adequate for simulations that have a simple (linear) circuit attached.

For problems which involve a complicated external circuit, it is best to apply the two-level Newton algorithm to the nonlinear solve. This algorithm is described in detail in Keiter [15] and Mayaram [16]. While this algorithm has been implemented and exercised within **Xyce**, it is not part of the Release 2.1 version. To use this algorithm, the user will need to obtain a special "custom" build of **Xyce**, or wait until a future release.

15.10 Output and Visualization

Using the .PRINT Command

For simple plots (such as I-V curves), output results for **Xyce** can be generated with the .PRINT statement, which is described in detail in section 10.1. Figures 15.4 and 15.9 are examples of the kind of data that is produced with .PRINT statement netlist commands. These particular figures were plotted in Tecplot, but many other plotting programs would also have worked, including XDAMP [27].

Multi-dimensional Plots

Device simulation has visualization needs which go beyond that of conventional circuit simulation. Multi-dimensional perspective and/or contour plots are often desirable. **Xyce** is capable of outputting multi-dimensional plot data in several formats, including Tecplot, GnuPlot, and sgplot. Currently, the options for each of these formats can only enable or disable the output of files, and when enabled, a new file (or a new append to an existing file) will happen at every time step or DC sweep step. For long simulations, this may produce a prohibitive number of files. Currently, there is no equivalent to the .OPTIONS OUTPUT INITIAL_INTERVAL command, nor does the output of plot data currently use this command. Plot files are either output at every step or not at all.

For each type of plot file, the file is placed in the execution directory. Each individual device instance is given a unique file, or files, and the file names are derived from the name of the PDE device instance. The instance names provides the prefix, and the file type (tecplot, gnuplot, sgplot) determines the suffix.

Tecplot Data

Tecplot is a commercial plotting program from Amtec Engineering, Inc., and is the best choice for creating contour plots of spatially dependent data. All of the graphical examples in this chapter were created with Tecplot. (see Figures 15.7 and 15.8 for examples) The output of Tecplot files is enabled using the instance parameter, tecplotlevel=1. If set to zero, no tecplot files are output. If set to one, a separate tecplot file is output for each nonlinear solve. If set to two, a single tecplot file, which contains data for every nonlinear solve is created and is appended at the end of each solve.

By default `tecplotlevel` is set to one, meaning the code will, by default produce a separate Tecplot file for each nonlinear solve. The suffix for Tecplot data files is *.dat. Internally, the file is an ASCII text file. Tecplot does have a binary format, but **Xyce** has not yet been set up to use it.

Note that it is also possible to set `tecplotlevel=2`. Doing this will force **Xyce** to create one single tecplot file, and the data from each solve will be appended to this file as a separate zone. This makes it possible to use Tecplot to create animations.

Gnuplot Data

Gnuplot is an open source plotting program, which is available on most Linux/Unix platforms. The parameter for this type of output is `gnuplotlevel=1`. This type of output file is off (zero) by default, meaning no gnuplot files will be output. The suffix for Gnuplot files is *Gnu.dat. Like tecplot files, Gnuplot files are also in ASCII text format.

NOTE: Gnuplot will only work with structured Cartesian meshes. Externally created, unstructured meshes (even ones that appear Cartesian) cannot be plotted with Gnuplot.

Sgplot Data

Sgplot is the plotting program for the SGFramework [20]. The parameter for this type of output is `sgplotlevel=1`. This type of output file is off (zero) by default. The suffix for Sgplot data files is *.res. Internally this file is in binary format. Note that it is not a machine-independent file format.

Volume Averaged Data

Xyce can also output volume-averaged information for each PDE device. This is enabled by setting the instance parameter, `txtdatalevel=1`. It is off (zero) by default, meaning no text files with volume averaged data will be output.

This page is left intentionally blank

Bibliography

[1] Laurence Nagel and Ronald Rohrer. Computer analysis of nonlinear circuits, excluding radiation (cancer). *IEEE Journal of Solid-State Circuits*, sc-6(4):166–182, 1971.

[2] Tom Quarles. Spice3f5 users' guide. Technical report, University of California-Berkeley, Berkeley, California, 1994.

[3] Eric R. Keiter, Thomas V. Russo, Eric L. Rankin, Richard L. Schiek, Keith R. Santarelli, Heidi K. Thornquist, , Deborah A. Fixel, Todd S. Coffey, and Roger P. Pawlowski. Xyce parallel electronic simulator: Reference guide, version 4.1. Technical Report SAND2008-xxxx, Sandia National Laboratories, Albuquerque, NM, January 2008.

[4] Orcad PSpice User's Guide. Technical report, Orcad, Inc., 1998.

[5] *gEDA Project Home Page*. `http://www.geda.seul.org/` .

[6] A. S. Grove. *Physics and Technology of Semiconductor Devices*. John Wiley and Sons, Inc., 1967.

[7] H. A. Watts, E. R. Keiter, S. A. Hutchinson, and R. J. Hoekstra. Time integration for the Xyce parallel electronic simulator. In *ISCAS 01*, October 2000.

[8] K.E. Brenan, S.L Campbell, and L.R. Petzold. *Numerical Solution of Initial-Value Problems in Differential-Algebraic Equations*. Society for Industrial and Applied Mathmatics, Philadelphia, 1996.

[9] Ljiljunu Trujkovit, Robert C. Melville, and Sun-Chin Fang. Passivity and no-gain properties establish global convergence of a homotopy method for dc operating points. *Proceedings - IEEE International Symposium on Circuits and Systems*, 2:914–917, 1990.

[10] Robert C. Melville, Ljiljana Trajkovic, San-Chin Fang, and Layne T. Watson. Artificial parameter homotopy methods for the dc operating point problem. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 12(6):861–877, 1993.

[11] J. Roychowdhury. *Private Communication*, 2003.

[12] H. G. Brachtendorf and R. Laur. On consistent initial conditions for circuit daes with higher index. *IEEE Transactions on Circuits and Systems-I: Fundamental Theory and Applications*, 48(5):606–612, 2001.

[13] T. Mei T. Coffey S. Hutchinson and J. Roychowdhury. Robust, stable time-domain methods for solving mpdes of fast/slow systems. In *Proc. IEEE Design Automation Conference*, 2004.

[14] Erik Boman, Karen Device, Robert Heaphy, Bruce Hendrickson, William F. Mitchell, Matthew St. John, and Courtenay Vaughan. *Zoltan: Data-Management Services for Parallel Applications:User's Guide*. `http://www.cs.sandia.gov/Zoltan/Zoltan.html`, 2004.

[15] Eric R. Keiter, Scott A. Hutchinson, Robert J. Hoekstra, Eric L. Rankin, Thomas V. Russo, and Lon J. Waters. Computational algorithms for device-circuit coupling. Technical Report SAND2003-0080, Sandia National Laboratories, Albuquerque, NM, January 2003.

[16] Kartikeya Mayaram and Donald O. Pederson. Coupling algorithms for mixed-level circuit and device simulation. *IEEE Transactions on Computer Aided Design*, II(8):1003–1012, 1992.

[17] David A. Johns and Ken Martin. *Analog Integrated Circuit Design*. John Wiley & Sons, Inc., 1997.

[18] Z. Yu, D. CHen, L. So, , and R. W. Dutton. Pisces-2et–two dimensional device simulation for silicon and heterostructures. Technical report, Stanford University, 1994.

[19] Davinci User's Manual. Technical report, TCAD Business Unit, Avanti! Corporation, 1998.

[20] Kevin M. Kramer and W. Nicholas G. Hitchon. *Semiconductor Devices: A Simulation Approach*. Prentice-Hall, Upper Saddle River, New Jersey, 1997.

[21] S. Selberherr. *Analysis and Simulation of Semiconductor Devices*. Springer-Verlag, New York, 1984.

[22] Eric R. Keiter, Scott A. Hutchinson, Robert J. Hoekstra, Eric L. Rankin, Thomas V. Russo, and Lon J. Waters. Computational algorithms for device-circuit coupling. Technical Report SAND2003-0080, Sandia National Laboratories, Albuquerque, NM, January 2003.

[23] *CUBIT Mesh Generation Toolsuite*. `http://sass1693.sandia.gov/cubit/`.

[24] N.D. Arora, J.R. Hauser, and D.J. Roulston. Electron and hole mobilities in silicon as a function of concentration and temperature. *IEEE Transactions on Electron Devices*, ED-29:292–295, 1982.

[25] D.M. Caughey and R.E. Thomas. Carrier mobilities in silicon empirically related to doping and field. *Proc. IEEE*, 55:2192–2193, 1967.

[26] J.M. Dorkel and Ph. Leturq. Carrier mobilities in silicon semi-empirically related to temperature, doping, and injection level. *Solid-State Electronics*, 24(9):821–825, 1981.

[27] *XDAMP Graphical User Interface*. `http://www.cs.sandia.gov/esimtools/xdamp.html` .

This page is left intentionally blank

Index

DISTRIBUTION:

1 Steven P. Castillo
Klipsch School of Electrical and Computer Engineering
New Mexico State University
Box 3-o
Las Cruces, NM 88003

1 Kwong T. Ng
Klipsch School of Electrical and Computer Engineering
New Mexico State University
Box 3-o
Las Cruces, NM 88003

1 Nick Hitchon
Electrical and Computer Engineering
University of Wisconsin
1415 Engineering Drive
Madison, WI 53706

1 Wendland Beezhold
Idaho Accelerator Center
1500 Alvin Ricken Drive
Pocatello, Idaho 83201

1 Linda Petzold
Department of Computer Science
University of California, Santa Barbara
Santa Barbara, CA 93106-5070

1 Jaijeet Roychowdhury
Department of Electrical Engineering and Computer Science (EECS) 545E Cory Hall
Berkeley, CA 94720-1770

1 C.-J. Richard Shi
VLSI and Electronic Design Automation
210 EE/CSE Bldg.
Box 352500
University of Washington
Seattle, WA 98195

1 Homer F. Walker
WPI Mathematical Sciences
100 Institute Road
Worcester, MA 01609

1 Dan Yergeau
CISX 334
Via Ortega
Stanford, CA 94305-4075

1 Tim Davis
P.O. Box 116120
University of Florida
Gainesville, FL 32611-6120

1 MS 0104
Thomas Bickel, 01200

1 MS 0139
Martin Pilch, 01221

1 MS 0139
Paul Yarrington, 01220

1 MS 0316
Joseph P. Castro, 01437

1 MS 0316
Deborah Fixel, 01437

1 MS 0316
Gary Hennigan, 01437

1 MS 0316
Robert J. Hoekstra, 01437

10 MS 0316
Eric R. Keiter, 01437

1 MS 0316
Paul Lin, 01437

1 MS 0316
Eric L. Rankin, 01437

1 MS 0316
Thomas V. Russo, 01437

1 MS 0316
Keith Santarelli, 01437

1 MS 0316
Richard Schiek, 01437

1 MS 0316
John N. Shadid, 01437

1 MS 0316
Heidi K. Thornquist, 01437

1 MS 0321
James Perry, 01400

1 MS 0341
Robert Brocato, 01711

1 MS 0341
Steven D. Wix, 01734

1 MS 0341
Carolyn Bogdan, 01734

1 MS 0348
John Dye, 05351

1 MS 0348
George R. Laguna, 05353

1 MS 0348
Perry Molley, 05351

1 MS 0350
Greg Lyons, 02625

1 MS 0350
Martin Stevenson, 02625

1 MS 0351
Richard Stulen, 01000

1 MS 0352
Charles E. Hembree, 17311

1 MS 0370
Tim Trucano, 01411

1 MS 0380
David Womble, 01540

1 MS 0384
Arthur Ratzel, 01500

1 MS 0405
Thomas D. Brown, 12332

1 MS 0405
Ben Long, 12346

1 MS 0405
Glenn L. Rice, 12346

1 MS 0406
Jason Dimkoff, 05712

1 MS 0428
Todd R. Jones, 12330

1 MS 0431
Leonard Lorence, 00511

1 MS 0431
Ronald Sikorski, 00512

1 MS 0447
Matthew T. Kerschen, 02111

1 MS 0457
Stephen Rottler, 02000

1 MS 0509
Mike Knoll, 05330

1 MS 0519
Scott A. Hutchinson, 05349

1 MS 0521
Edward Boucheron, 02617

1 MS 0525
Roger F. Billau, 01734

1 MS 0525
Mike Deveney, 01734

1 MS 0525
Raymond B. Heath, 01734

1 MS 0525
Albert Nunez, 01734

1 MS 0525
Paul V. Plunkett, 01734

1 MS 0525
Regina Schells, 01734

1 MS 0611
Siviengxay Limary, 02958

1 MS 0638
Charles Michael Williamson, 12344

1 MS 0801
Robert Leland, 09300

1 MS 0807
David N. Shirley, 09326

1 MS 0807
Philip M. Campbell, 09326

1 MS 0823
John D. Zepper, 09320

1 MS 0828
Anthony A. Giunta, 01544

1 MS 0972
Stephen E. Lott, 05572

1 MS 0982
Barbara Wampler, 05733

1 MS 1002
Philip Heermann, 06470

1 MS 1071
Dahlon Chu, 01730

1 MS 1079
Gilbert Herrera, 01700

1 MS 1083
Paul E. Dodd, 17311

1 MS 1116
Stephen L. Brandon, 04245

1 MS 1137
Greg D. Valdez, 06223

1 MS 1138
Rebecca Arnold, 06225

1 MS 1138
Mark A. Gonzales, 06225

1 MS 1138
Harvey C. Ogden, 06225

1 MS 1152
Mark L. Kiefer, 01652

1 MS 1153
Larry D. Bacon, 05443

1 MS 1179
Brian Franke, 01341

1 MS 1219
David E. Beutler, 05923

1 MS 1221
Marion Scott, 05600

1 MS 1231
Steven N. Kempka, 05223

1 MS 1316
Elebeoba May, 01412

1 MS 1316
Scott Mitchell, 01415

1 MS 1316
Mark D. Rintoul, 01412

1 MS 1318
Karen Devine, 01416

1 MS 1318
Mike Eldred, 01411

1 MS 1318
Bruce Hendrickson, 01415

1 MS 1318
Roger Pawlowski, 01414

1 MS 1318
Andrew Salinger, 01414

1 MS 1318
Bart van Bloemen Waanders, 01414

1 MS 1319
Jim Ang, 01422

1 MS 1319
Robert Benner, 01422

1 MS 1319
Doug Doerfler, 01422

1 MS 1319
Neil Pundit, 01420

1 MS 1320
Todd Coffey, 01414

1 MS 1320
David Day, 01414

1 MS 1320
Mike Heroux, 01416

1 MS 1320
James Willenbring, 01416

1 MS 1322
John Aidun, 01435

1 MS 1322
Sudip Dosanjh, 01420

1 MS 1322
Harry Hjalmarson, 01435

1 MS 1323
Pat Crossno, 01424

1 MS 1323
David Rogers, 01424

1 MS 1330
Scott Holswade, 05340

1 MS 1393
Doug Weiss, 02110

1 MS 1453
Jaime L. Moya, 02550

1 MS 9004
William P. Ballard, 08100

1 MS 9007
Rene L. Bierbaum, 08245

1 MS 9007
Brian E. Owens, 08245

1 MS 9102
Seung Choi, 08229

1 MS 9102
Rex Eastin, 08227

1 MS 9102
Kathryn R. Hughes, 08228

1 MS 9151
Leonard Napolitano, 08900

1 MS 9158
Kevin R. Long, 08961

1 MS 9158
Mei Ting, 01437

1 MS 9159
Tamara G. Kolda, 08962

1 MS 9402
Donna J. O'Connell, 08134

1 MS 0899
Technical Library, 9536 (electronic copy)